175 PTAS
mi computer
91
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
IMAGES 9
IMAGES 9
IMAGES 8
IMAGES 7
Graphic design handbook
MECANORMA
NIMBUS PC2
RM NIMBUS
RM NIMBUS
ZUCOR
Editorial Delta

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 91

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°. 08021 Barcelona
Tel. (93) 209 80 22

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 238510
Impreso en España-Printed in Spain-Octubre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.





Aprender a aprender

Para resolver muchos de los problemas que se presentan en el campo de la AI es preciso crear un sistema que aprenda por sí solo

Si, con el correr del tiempo, un sistema de ordenador ha perfeccionado su ejecución de una tarea determinada sin ser preprogramado, cabe afirmar que ha aprendido. Es importante destacar que ello supone un criterio acordado, o normalizado, en relación al cual se pueda medir al sistema. En ausencia de una forma acordada de evaluar el progreso, no tiene ningún sentido hablar de aprendizaje. Por consiguiente, un algoritmo de aprendizaje intenta cumplir uno o más de los siguientes cometidos:

- Cubrir una gama más amplia de problemas.
- Ofrecer soluciones más exactas.
- Obtener respuestas a un costo inferior.
- Simplificar el conocimiento codificado.

El último punto da por sentado que la simplificación del conocimiento almacenado es valiosa, aun cuando no resulte en perfeccionar el rendimiento de la tarea del ordenador. Ello puede ser así si el sistema comienza con un conjunto de reglas .y acaba con otro conjunto, igualmente eficaz, que resulte ser más comprensible para nosotros.

El aprendizaje de la máquina se puede aplicar en muchos campos; pero los sistemas de aprendizaje de mayor éxito se han aplicado a problemas de clasificación.

Ian McKinnell

Inteligencia crítica
Los psicólogos han sugerido que el niño aprende mediante la formación de estructuras de reglas, conocidas como *esquemas*, a menudo mediante un proceso de ensayo y error, comprobando hipótesis (las *reglas*) y conservando aquellas que ofrecen resultados correctos. En muchos casos el proceso de aprendizaje lo dirige alguien que posee un conocimiento sobre los resultados ideales (el *crítico*). El crítico (o maestro) ayuda al niño (el *aprendiz*) a evaluar y refinar su conjunto interno de reglas, que lo ayudará a perfeccionar su rendimiento ante una tarea determinada. Los sistemas de aprendizaje AI intentan reproducir este proceso mediante la creación de una base de conocimientos que se pueda utilizar junto con un conjunto de aprendizaje de ejemplos y un método para evaluar el rendimiento de cada regla

El objetivo de un sistema de este tipo consiste en tomar un dato de entrada y clasificarlo, identificarlo o interpretarlo de alguna manera.

Se han probado diversos procedimientos para obtener mejoras automáticas del rendimiento, siendo Arthur Samuel, con sus clásicos estudios del aprendizaje de la máquina mediante el juego de damas, el pionero de dos de los métodos más simples: el *aprendizaje de rutina* y el *ajuste de parámetros*.

El *ajuste de parámetros*, un método en el cual los coeficientes y los parámetros del programa se ajustan repetidamente al objeto de mejorar el rendimiento, es simplemente una especie de técnica de optimización. Este método se ha estudiado de forma exhaustiva en la literatura sobre matemática aplicada y, por lo tanto, está relativamente bien comprendido. El *aprendizaje de rutina* es completamente "no creativo", y puramente una técnica de compresión del almacenamiento, mientras que los métodos que consideramos aquí son capaces de generalizar y, por tanto, aprender la respuesta apropiada en una situación hasta entonces desconocida.

Todo sistema diseñado para crear nuevos conocimientos y, por tanto, mejorar su rendimiento, debe incluir los siguientes componentes fundamentales:

- Un conjunto de estructuras de datos que represente el actual nivel de pericia del sistema (las *reglas*).
- Un algoritmo de tarea (el *ejecutante*) que utilice las reglas para dirigir la actividad de resolución del problema.
- Un módulo de realimentación (el *crítico*) que compare los resultados actuales con los objetivos deseados.
- El mecanismo de aprendizaje propiamente dicho (el *aprendiz*) que emplee la realimentación del crítico para corregir las reglas.

El método de representación elegido para codificar el conocimiento del sistema es al menos tan importante como los detalles del algoritmo de aprendizaje. En consecuencia, antes de construir un sistema de aprendizaje, es esencial asegurar que el lenguaje de descripción empleado con el sistema sea capaz de expresar las clases de distinciones que serán necesarias, problema nada trivial.

Suponiendo que podamos resolver el problema de idear un lenguaje de descripción adecuado, nos queda el problema de automatizar la generación de descripciones precisas mediante el mismo.

Una forma de enfocar el problema es hacer como si se tratara de una búsqueda a través de todas las descripciones posibles con el fin de hallar aquellas que sean útiles en un contexto dado. Un sistema de AI que aprenda a clasificar puede comenzar con una serie de parámetros que se utilizarán para efectuar una clasificación. En la fase de aprendizaje, el sistema podrá generar y evaluar otras descripciones mediante la combinación, de una u otra manera, de los parámetros originales, conservando las descripciones que sean de ayuda para una clasificación correcta y descartando las que no sirvan. La cantidad de descripciones generadas que sean válidas sintácticamente puede ser astronómica, y, cuanto más expresivo sea el lenguaje de descripción, más explosivo será este problema combinatorio.

Es evidente que se debe hallar alguna manera de dirigir la búsqueda ignorando al mismo tiempo la inmensa mayoría de descripciones potenciales que sean irrelevantes.

Numerosos métodos han funcionado bien como casos de aprendizaje "libres de ruido", en los cuales las clasificaciones están bien definidas, con pocas áreas oscuras entre las clases. Sin embargo, tratar con datos "ruidosos" es un problema que plantea más desafíos. Veamos ahora un procedimiento comparativamente simple que parece funcionar bastante bien en este segundo caso.

BEAGLE (*Biological Evolutionary Algorithm Generating Logical Expressions*: algoritmo de evolución biológica que genera expresiones lógicas) es un sistema de ordenador que produce reglas de decisión mediante la inducción a partir de una base de datos. Como tal, direcciona el problema (con frecuencia eludido) de saber de dónde provienen las reglas de un sistema basado en reglas. BEAGLE trabaja según el principio de la "selección natural", en el que las reglas que no se adaptan a los datos se suprimen y se sustituyen por "mutaciones" de reglas mejores, o por reglas nuevas creadas mediante la unión de dos reglas mejor adaptadas. Las reglas son expresiones booleanas representadas mediante estructuras arborescentes.

El software original estaba compuesto por dos programas en PASCAL: HERB (*Heuristic Evolutionary Rule Breeder*: creador de reglas evolucionistas heurísticas) y LEAF (*Logical Evaluator And Forecaster*: evaluador y pronosticador lógico).

HERB exige que el usuario cree tres archivos de entrada: uno de datos, otro de resultados finales y un archivo de reglas viejo, que puede estar vacío. Produce como salida un archivo de reglas nuevo, que es al menos tan bueno como el viejo. El archivo de datos contiene el conjunto de aprendizaje, para el cual se conoce la pertenencia a la categoría correcta. Asimismo, el usuario ha de llenar una matriz de resultado final, que define el valor o costo de clasificaciones correctas e incorrectas.

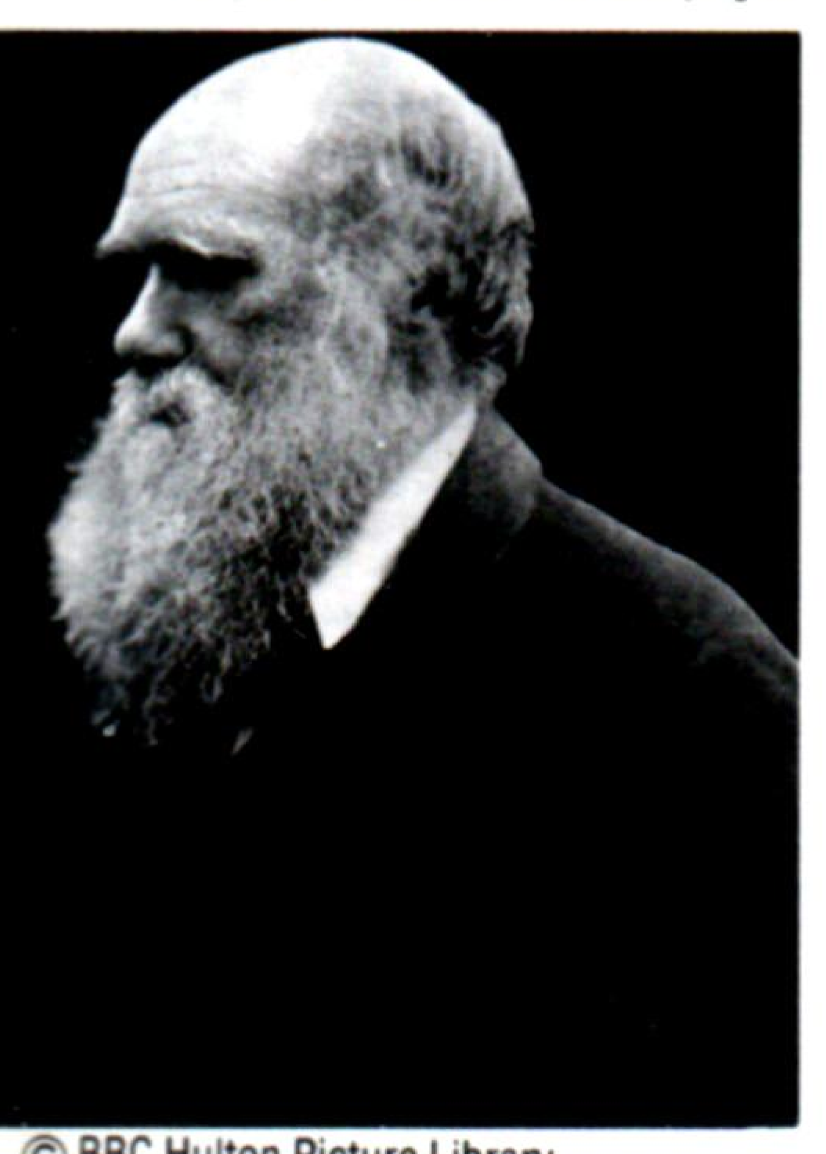
LEAF es más simple que HERB. Toma sencillamente un archivo de datos del mismo formato que el conjunto de aprendizaje y ejecuta sobre él un archivo de reglas. Se le puede solicitar que imprima, entre otras cosas, una lista ordenada de elementos del archivo de datos de aquellos que tengan mayores probabilidades de encajar en una clase dada (tal como la defina el archivo de reglas) hasta aquellos que menos probabilidades tengan de encajar en la clase.

El algoritmo BEAGLE consiste en repetir el procedimiento siguiente durante un cierto número de "generaciones" (una generación es una pasada completa a través de los datos de aprendizaje).

1. Evaluar cada regla en cada muestra de acuerdo a la matriz de resultado final, dándole una prima a las reglas más cortas.
2. Colocar las reglas por orden descendente de méritos y suprimir la mitad inferior.
3. Reemplazar las reglas "muertas" aplicándole un procedimiento de emparejamiento a un par de supervivientes elegidas al azar, recombinando, por tanto, porciones de reglas buenas.
4. Transformar unas pocas reglas elegidas al azar (pero no la superior) y aplicarles a todas las reglas nuevas un procedimiento TIDY, preparado ya para la próxima generación.



Charles Darwin (1809-1882)
Las teorías de Darwin sobre la evolución sostienen que las especies se adaptan y se perfeccionan a sí mismas en respuesta a su medio ambiente mediante la selección natural, sobreviviendo sólo los miembros más fuertes y mejor adaptados para criar la siguiente generación y transmitirle sus características. Esta idea se ha empleado con total éxito en el campo del aprendizaje de la máquina, permitiendo que los sistemas de AI mejoren su rendimiento en la clasificación de problemas mediante el desarrollo de conjuntos de reglas de clasificación. No es mera coincidencia el hecho de que uno de tales sistemas se denomine BEAGLE, nombre del barco de Darwin en su famosa expedición a las islas Galápagos



© BBC Hulton Picture Library

El procedimiento TIDY reduce ciertas redundancias sintácticas (en inglés) que se puedan haber generado, como dobles negaciones y expresiones constantes, dejando el árbol de reglas "podado" esencialmente con el mismo conjunto de reglas, con la excepción que estará expresado de forma más sucinta. Este método para la generación de reglas nuevas se basa en las ideas darwinianas sobre la evolución: para criar a la nueva generación sólo sobreviven las reglas más aptas.

Informe meteorológico

Un ejemplo de sistema de aprendizaje es el que utiliza el registro del tiempo de un día para pronosticar si lloverá al día siguiente. Para determinar esto la máquina debe "aprender" una regla buena. El sistema puede empezar con una serie de características sobre el tiempo: precipitaciones, intensidad del sol, velocidad máxima del viento y presión del aire (todo ello medido por la tarde, pongamos por caso). Para poder aprender las reglas que permitirán que la máquina formule pronósticos adecuados, se debe reunir un conjunto de registros de datos. Por cada registro se sabrá si lloverá al día siguiente. Un registro típico sería:

Lluvia	0
Sol	7.2
Viento	22
Presión	1017

El sistema debe producir reglas (tal vez, al principio, de forma aleatoria) y utilizar el aprendizaje para comprobar su validez. Las reglas se pueden retener como componentes, tales como:

Características	Viento
Constantes	10
Operadores de comparación	> o <
Operadores lógicos	AND, OR y NOT

y éstos combinarse en expresiones como:

sol <4 AND presión <1000

Tras cada pasada de aprendizaje se pueden conservar las reglas buenas y desechar las malas, sustituyendo éstas por nuevas reglas "criadas" con partes de las reglas buenas restantes. Desarrollando reglas que mejoran su rendimiento a cada pasada de entrenamiento, la máquina debe mejorar su capacidad para pronosticar el tiempo de mañana

Generaciones de reglas

GENERACION
PADRE
HIJOS
NIETOS
BIZNIETOS

Lluvia > 1
Negar
Disminuir constante
Aumentar constante
Generalización
NOT (Lluvia > 1)
Lluvia > 0
Lluvia > 2
Lluvia > 1 OR SOL > 1
Nudo ancestral regenerado
Disminuir constante
Aumentar constante
Generalizar
Lluvia > 1
Lluvia > 3
Lluvia > 2 OR presión >999
Generalizar
Especializar
Lluvia > 3 OR viento > 10
Lluvia > 3 AND viento > 10

Izen.

A partir de una descripción inicial, como "lluvia>1", se puede generar una serie de nuevas descripciones transformando la sentencia original o combinándola con otras sentencias. De esta forma se puede producir un árbol de descendientes a partir de una única descripción, y considerar el proceso de aprendizaje como una búsqueda de descripciones adecuadas a través del árbol. Dado que las aplicaciones repetidas de una transformación pueden llevar a la regeneración de un nudo ancestral (como vemos en el ejemplo), el espacio de trabajo implícito es en realidad una red

Gabrielle Izen

Réplica perfecta

Al impartir la instrucción SYSGEN, el CP/M se copia a sí mismo de un disco a otro

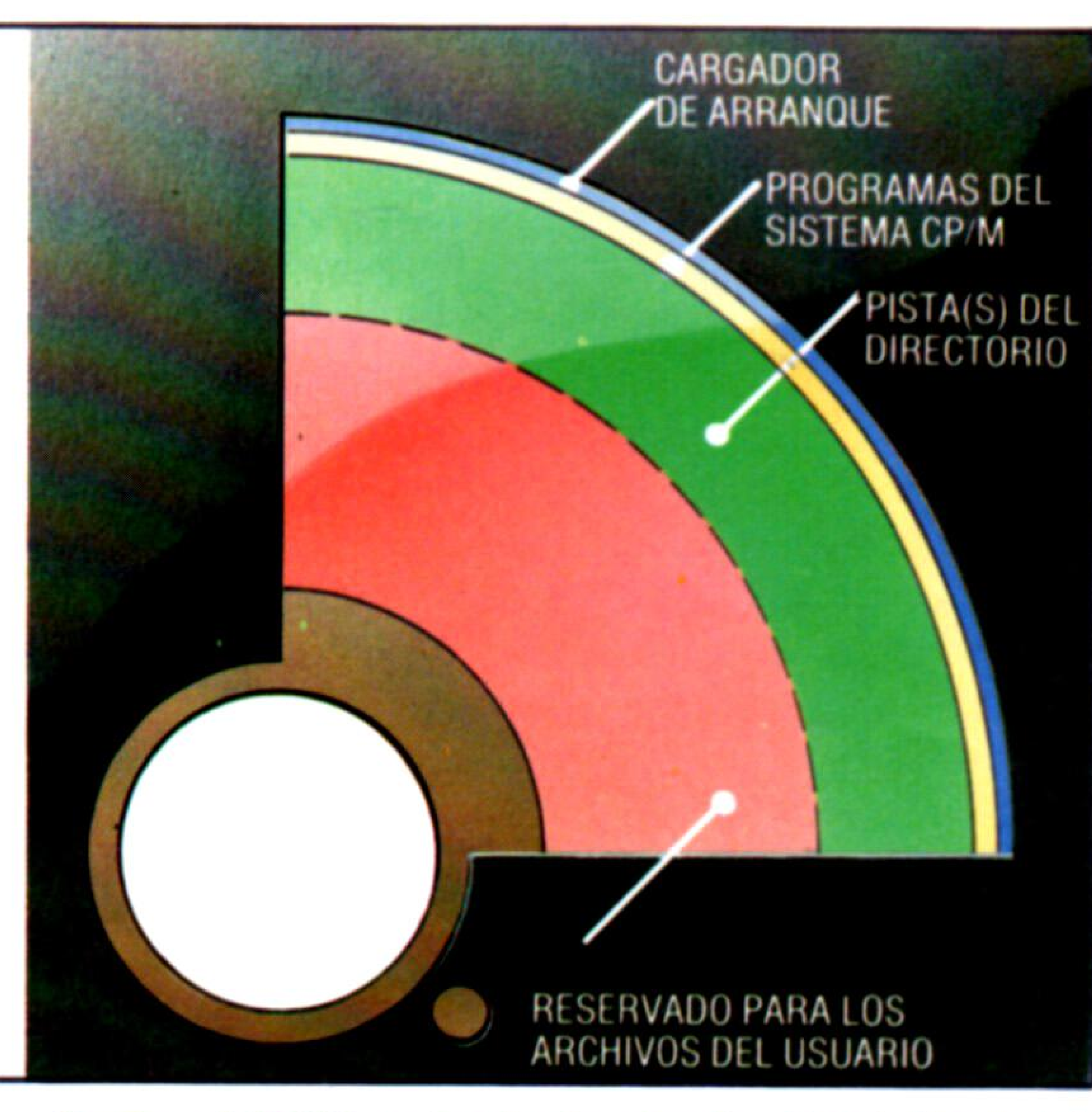
Hasta ahora en esta serie hemos venido examinando la gama de instrucciones disponibles en CP/M y ya estamos en condiciones de llevar a cabo todas las funciones que se podrían necesitar en casa o en la oficina. Podemos cargar, editar, guardar y transferir los diversos tipos de archivos que están definidos por el CP/M, así como examinar sus propiedades. No obstante, hay una función vital que no hemos realizado hasta el momento: a saber, copiar el propio CP/M.

Como es obvio, es sumamente importante poder transferir el CP/M a otros discos; no sólo para proveernos de una copia de seguridad, sino también porque muchas aplicaciones requieren colocar en el disco una versión del CP/M especialmente instalada. Un ejemplo de ello es el Dr. LOGO en el Amstrad, que incorpora una versión del CP/M hecha a medida en la misma cara que el lenguaje que, a su vez, contiene procedimientos extras para editar y colorear.

Formando pistas

Si bien la mayor parte de un disco de datos CP/M está disponible para archivos del usuario, las pistas más exteriores está reservadas para que las utilice el propio OS. La pista cero retiene el cargador de arranque, que carga automáticamente el CP/M tras el encendido y el reset, mientras que las pistas uno y dos retienen el sistema CP/M propiamente dicho. Tras ellas están las pistas del directorio, que retienen los FCB, que muestran exactamente dónde están situados los registros de que consta cada archivo

Kevin Jones



Copiar el CP/M es bastante simple. En el disco del sistema se proporciona una instrucción transitoria llamada SYSGEN; al ejecutarla, el programa le presentará una serie de preguntas, según las características del sistema, que se pueden utilizar para transferir el CP/M al disco de destino.

Esto suena muy fácil. Sin embargo, la operación plantea algunas cuestiones interesantes sobre la naturaleza del CP/M. ¿Por qué necesitamos una instrucción especial? ¿Por qué en un listado del DIRectorio no aparecen ni el CP/M ni sus instrucciones residentes (incorporadas)? Y, además, ¿cómo lee realmente el CP/M un directorio? Para responder a estas preguntas, centrémonos en la construcción del CP/M.

El OS que se carga en el ordenador se puede dividir en tres partes, cada una de las cuales maneja una parte determinada de la operación de disco. La parte con la que usted estará más familiarizado es el CCP (*Console Command Processor*, procesador de instrucciones de consola), que parece natural puesto que se trata de la "fachada" del sistema, el área que envía información al monitor de video e interpreta las instrucciones que se digitan en el teclado.

Sin embargo, hace muchas más cosas entre bastidores. Cuando se digita información y se entra por el teclado, el CCP da por sentado que la información representa instrucciones y, en consecuencia, éstas se transfieren al área de buffer del CCP a través de otra parte del sistema denominada BIOS (*Basic Input/Output System*: sistema básico de entrada y salida). El CCP realiza, entonces, una comprobación de las instrucciones residentes para ver si su instrucción corresponde a alguna de ellas. De ser así, esa instrucción se ejecutará de inmediato. Si la instrucción no es ninguna de las "incorporadas", el CCP dará por sentado que es transitoria y la buscará en el disco del sistema. Una vez localizada, se cargará en la memoria y se ejecutará. Si el CCP no consigue encontrar la instrucción, generará, por supuesto, un mensaje de error *"file not found"* (archivo no hallado). En el caso del CP/M, el CCP visualizará el nombre de la instrucción en letras mayúsculas, seguidas por un signo de interrogación. La causa de que la instrucción se visualice en mayúsculas es que una de las funciones del CCP es convertir todas las instrucciones en minúsculas a mayúsculas, para adecuarlas al sistema de denominación de archivos del CP/M

Sistema básico de entrada/salida

El BIOS es un conjunto de rutinas (conocidas como activadoras de periféricos) que manipulan todas las funciones de entrada/salida del CP/M. Su finalidad principal consiste en gestionar los periféricos que estén conectados en cada momento al sistema (incluyendo el teclado y la pantalla: el CCP se limita a enviar e interpretar información a través del BIOS). El sistema utiliza parámetros proporcionados por el CCP. Si usted deseara "diseñar a medida" el CP/M para ejecutarlo en otra máquina, es el BIOS lo que tendría que alterar para que el programa se pudiera ejecutar de forma correcta. El motivo de ello (aparte de que el CP/M tenga ciertas exigencias de hardware) es que el método de administración de periféricos puede experimentar grandes variaciones entre máquinas diferentes.

La tercera parte del CP/M es la sección que se ocupa directamente de la gestión del disco. Esta sección se denomina BDOS (*Basic Disk Operating System*: sistema operativo básico de disco). Éste es el componente menos "visible" del CP/M y, sin él, el CP/M sería incapaz de alcanzar su objetivo fundamental. BDOS es el software que administra los archivos en disco, asigna

áreas para almacenamiento y mantiene al día el directorio. A continuación, veamos cómo se distribuye el espacio de almacenamiento en un disco flexible.

Sectores "soft" y "hard"

Desde el punto de vista del hardware, un disco está dividido en 40 pistas. Las versiones modernas del CP/M también tendrán las pistas divididas en *sectores soft* (así llamados porque es el software el que determina los sectores). Los tipos más antiguos de sistemas de disco flexible subdividían las pistas en *sectores hard*, que eran leídos mecánicamente por medio de hardware.

La mayoría de las pistas de un disco flexible se utilizan para almacenar información del usuario; no obstante, hay tres pistas (en un disco de 5 ¼ pulgadas) reservadas para que las utilice el CP/M. Estas pistas exteriores (numeradas del 0 al 2) contienen el breve programa *cargador del arranque* (también conocido como *monitor residente*), que permite que el CP/M se cargue a sí mismo en el ordenador sin que haya ningún sistema operativo de disco presente en memoria. Estas pistas también contienen el sistema CP/M propiamente dicho. Es la información de estas tres pistas la que se instala en el disco mediante la instrucción SYSGEN.

Los sectores (a los que se alude en términos CP/M como *registros*) contienen 128 bytes de información. Se pueden agrupar registros entre sí para formar una *unidad*; una unidad puede contener hasta 128 registros. Un archivo CP/M puede retener hasta 16 de estas unidades. Por lo tanto, un archivo se compone de un máximo de 128 por 128 por 16 bytes (256 Kbytes).

A menos que el archivo sea verdaderamente muy pequeño, no se lo podrá retener en un solo registro. Tampoco será posible, en términos prácticos, acomodar todos los registros de un archivo por orden secuencial en disco. Por consiguiente, un sistema operativo de disco debe poseer algún método para saber qué sectores de un disco se están utilizando para cada uno de los archivos particulares.

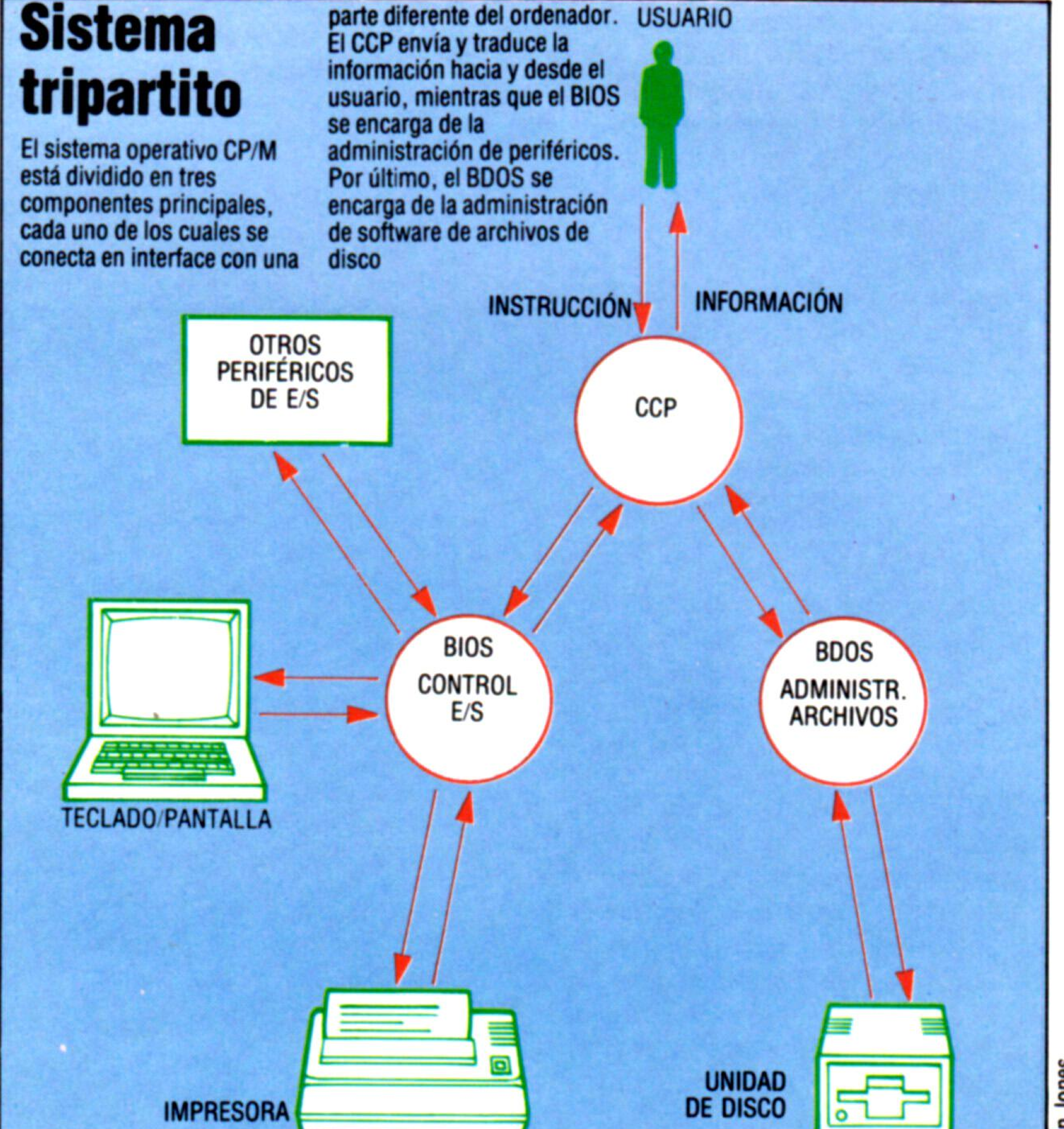
En CP/M, esta información se conoce como FCB (*File Control Block*: bloque de control de archivos). En un disco de 5 ¼ pulgadas, estos FCB están retenidos en la cuarta pista (que se conoce como pista del directorio). Este método de indexación de archivos es leído, escrito, alterado y administrado por el BDOS. Un FCB se compone de hasta 33 bytes de información y retiene todos los datos necesarios para posibilitar que el BDOS identifique y localice cualquier archivo en un disco.

El FCB se compone, en primer lugar, del "tipo de entrada", un único byte que retiene la identidad de la unidad de disco en la cual reside actualmente el disco que contiene el archivo. Los ocho bytes siguientes contienen el nombre del archivo seguido por tres bytes que están reservados para la extensión. Los bytes del 12 al 14 se utilizan para retener la extensión total del archivo, mientras que el byte 15 retiene el contador de registros (que indica cuántos registros tiene el archivo). Los 15 bytes siguientes contienen el mapa de asignación de disco del archivo. Por último, el último byte retendrá el siguiente número de registro a acceder (en caso de que el archivo ocupe más de los 16 K que se pueden describir mediante un solo FCB).

Cuando se ha abierto un archivo mediante la ejecución de una instrucción, el CCP crea una versión del FCB en el área de memoria normalmente reservada para instrucciones transitorias. La información extra acerca del archivo (como su longitud, su cantidad de registros y su asignación de sectores) la proporcionará el BDOS cuando encuentre un archivo cuyo nombre concuerde con el creado por el CCP. La razón de colocar un FCB en la memoria es para que el BDOS pueda actualizar rápidamente los datos retenidos en él mientras las instrucciones del CP/M manipulan el archivo. Finalizadas estas operaciones, el BDOS escribirá en disco la versión final del FCB, teniendo en cuenta la longitud de registros y asignación de sectores alterados.



Ahora podemos retroceder un paso y resumir cómo recupera el CP/M sus archivos. Cuando el CCP recibe una instrucción desde el terminal, interpreta la primera palabra como la instrucción. Si la instrucción no está residente en la memoria, el CCP enviará el nombre del archivo al BDOS a través del sistema de "mensajería" BIOS. El BDOS buscará entonces la pista del directorio del disco hasta hallar una pareja para el nombre de archivo en uno de los FCB. Si el archivo es de instrucciones (COM), el BDOS examinará el mapa de asignación del disco para descubrir qué sectores del disco están ocupados por el archivo; una vez localizados, los copiará en la memoria, en cuyo punto se ejecutará la instrucción.



Sistema tripartito
El sistema operativo CP/M está dividido en tres componentes principales, cada uno de los cuales se conecta en interface con una parte diferente del ordenador. El CCP envía y traduce la información hacia y desde el usuario, mientras que el BIOS se encarga de la administración de periféricos. Por último, el BDOS se encarga de la administración de software de archivos de disco



Esto puede significar que se requiera el BDOS para hallar otro archivo sobre el cual actuar, así como para buscarlo y cargarlo en la memoria. Cuando se carga un archivo en la memoria, el BDOS también proporciona los datos de FCB adecuados a una versión vacía de un FCB que haya reparado el CCP. Este FCB se actualizará a medida que avance la ejecución. Al final del programa de instrucciones, la versión actualizada se volverá a copiar en el disco mediante el BDOS.

Efectos laterales

Veamos cómo el PROLOG incorpora "efectos laterales" y varias características "extralógicas"

El PROLOG, como hemos visto, es un lenguaje esencialmente lógico, pero ciertas funciones que necesita llevar a cabo un lenguaje de programación, como leer y escribir archivos y efectuar cálculos aritméticos, son básicamente procesales y no se ajustan con comodidad a la lógica de predicado. Con el fin de hacer frente a estas funciones, el PROLOG posee una cantidad de predicados incorporados. Algunos de éstos operan en base a efectos laterales: write(Término), por ejemplo, triunfará siempre, pero, como efecto lateral, llevará a cabo alguna función útil (en este caso, escribir el valor de Término en el canal de salida actual).

En PROLOG, todas las entradas y salidas se manipulan a través de efectos laterales.

Escribir archivos en PROLOG es lo mismo que escribir en la pantalla. Un predicado, tell(Nombrearchivo)(*tell*: decir), le indica al PROLOG que envía toda futura salida al archivo especificado; told (dicho) cierra el archivo. De modo similar, see(Nombrearchivo) (*see*: ver) le dice al PROLOG que lea el archivo mencionado, y seen (visto) restablece el *status quo*. El PROLOG puede enviar códigos ASCII utilizando put(Código) (*put*: poner), mientras que get (obtener) lee valores ASCII.

Para efectuar aritmética, el PROLOG posee varios predicados incorporados que en realidad son funciones que evalúa el intérprete. Uno de tales predicados "evaluables" es is (es). Éste se utiliza de la siguiente manera:

```
C1 is C+1.
```

Que, en BASIC, equivale a:

```
C1=C+1.
```

Observe que C is C+1 siempre fracasará en PROLOG, mientras que el C=C+1 del BASIC funcionará siempre perfectamente. Esto se debe a que lenguajes como el BASIC, FORTRAN, PASCAL, ALGOL y C emplean la asignación destructiva. Es decir, al asignar el resultado de la expresión (a la derecha de una sentencia de asignación) a la variable (a la izquierda), todo valor que esté retenido en esa variable se sobreescribirá.

El PROLOG no utiliza la asignación de ninguna manera. En cambio, emplea la unificación. Esto es, una vez que se ha ejemplificado una variable, toda otra ocurrencia de la misma variable en la totalidad de la cláusula toma el mismo valor. Por tanto, decir que C is C+1 sería como tratar de darle a C dos valores diferentes al mismo tiempo.

El predicado incorporado not (no) se utiliza para introducir información negativa y se define de modo que not(X) será verdadero si X es falso y viceversa, igual que el NOT booleano del BASIC. Para que not(X) sea verdadero, el PROLOG debe primero demostrar que X es falso. Esto lo hace de una forma bastante candorosa: ¡una cosa es falsa si no se puede demostrar que sea verdadera! Imaginemos que tenemos el siguiente programa en PROLOG:

Versión en PROLOG estándar:

```
lenguaje(pascal,difícil)
lenguaje(cobol,difícil)
lenguaje(basic,sencillo)
lenguaje(prolog,sencillo)
```

Versión en MICRO-PROLOG:

```
(lenguaje Pascal difícil)
(lenguaje Cobol difícil)
(lenguaje BASIC sencillo)
(lenguaje Prolog sencillo)
```

que lista unos pocos hechos acerca de la facilidad que entraña la programación en diversos lenguajes. Podríamos formular preguntas tales como:

```
lenguaje(basic,DIFICULTAD).
```

donde DIFICULTAD es una variable, y el PROLOG responderá:

```
DIFICULTAD=sencillo.
```

Pero si preguntamos:

```
lenguaje(basic,fácil).
```

el PROLOG nos responde no. Ello se debe a que el objetivo que hemos planteado no se puede demostrar porque no está ni explícitamente enunciado en la base de datos, ni se puede derivar de otros hechos y reglas. El PROLOG, por supuesto, no comprende el significado de las palabras, de modo que no puede saber que sencillo y fácil significan lo mismo.

Esta idea de la "negación como fallo" es uno de los puntos más débiles del PROLOG y se debe al método de prueba lógica que utiliza el intérprete. Éste es llamado *resolución* y no puede manipular información negativa directamente. Por ejemplo, consideremos esta regla:

```
not(A):-B,C,D
```

(que afirma que X no es el caso si B, C y D son verdaderos). Esto no se puede escribir en PROLOG porque en el encabezamiento de una cláusula no puede haber una negación.

Características extralógicas

El PROLOG no es un lenguaje puramente declarativo. Ya hemos visto cómo el ordenamiento de las cláusulas en el código fuente puede influir en la forma en que funciona el programa, en especial en los procedimientos recursivos. De hecho, la pureza de su lógica se puede diluir aún más mediante el empleo de las llamadas características "extralógicas", fail (fallo) y cut (corte).

La primera de éstas, fail, se puede explicar fácilmente. Se trata de un predicado que fracasará siempre que se lo encuentre. ¿Por qué necesitamos a fail? ¡La razón principal es la de mantener trabajando al PROLOG! Si estamos utilizando nuestro programa de lenguajes y preguntamos:

```
lenguaje(Lang,sencillo).
```

el PROLOG responderá:

```
Lang=basic
```

y se detendrá. Encuentra la primera cláusula de su base de datos que concuerda con el objetivo que hemos planteado y, habiendo demostrado el objetivo, ya no busca más. Pero puede que queramos obtener una lista de todos los lenguajes sencillos. Podemos obligar al PROLOG a producirla si añadimos una regla como:

Versión en PROLOG estándar:

```
listasencillo:-lenguaje(Lang,sencillo),
  write(Lang),nl,fail
```

Versión en MICRO-PROLOG:

```
((listasencillo)(lenguaje X sencillo)
(PX)PP FAIL)
```



Ahora, si digitamos listasencillo, el PROLOG replicará:

```
basic
prolog
no
```

Lo que sucedió es que el PROLOG intentó demostrar el objetivo listasencillo demostrando en primer lugar lenguaje(Lang,sencillo). Halló el hecho lenguaje(basic,sencillo) y estableció la variable Lang en el valor basic. Así se demostró el primer subobjetivo. Pasó al siguiente y, con Lang siendo basic, encontró write(basic). Puesto que write triunfa automáticamente (escribiendo basic como un efecto lateral), pasó al subobjetivo siguiente, que es fail. Éste, por supuesto, fracasó, de modo que el PROLOG retrocedió a write. Dado que al retroceder write (y los otros predicados evaluables) tampoco volvieron a triunfar, el PROLOG retrocedió aún más hasta lenguaje(Lang,sencillo) e intentó hallar otra forma de demostrar su veracidad.

La otra característica, cut (¡que se escribe !), también se utiliza para controlar el retroceso. Un cut es un predicado que siempre triunfa. No obstante, como efecto lateral, impide que el intérprete retroceda a través de él. Esto efectivamente congela cualquier opción sobre los valores de variables tomados previamente en un procedimiento. Asimismo, impide que se prueben cláusulas alternativas en el mismo procedimiento. Un uso importante de cut es, en conjunción con fail, asegurar que no se vuelve a probar otra vez una cláusula deliberadamente fallida. Podríamos definir un predicado faltante empleando ! y fail del siguiente modo:

Versión en PROLOG estándar:

```
faltante(Cláusula):–Cláusula,!,fail.
faltante(Cláusula).
```

Versión en MICRO-PROLOG:

```
((faltante X)(?X)/FAIL
(faltante X)
```

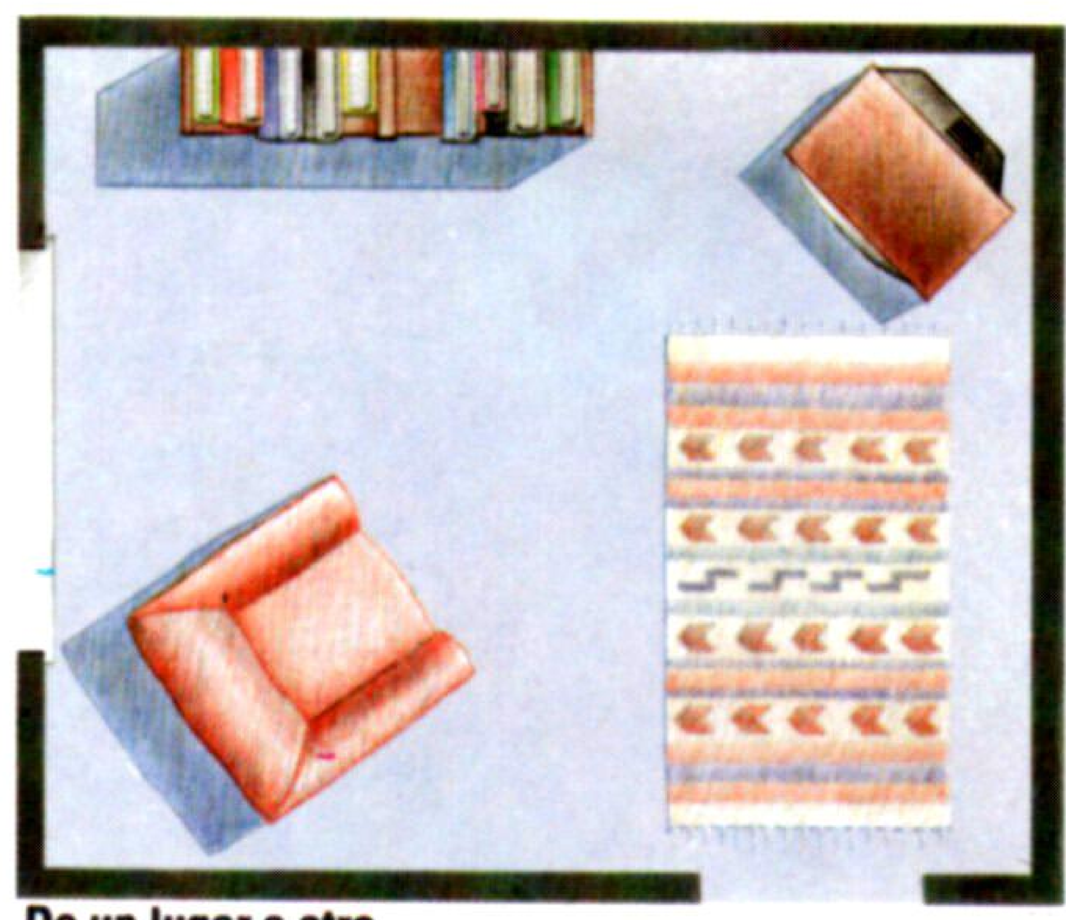
que es verdadero si la cláusula dada como su argumento no está incluida en la base de datos, y falso en caso contrario. Así, podríamos preguntar:

```
faltante(lenguaje(basic,sencillo)).
```

Para demostrar faltante(lenguaje(basic, sencillo)), debemos demostrar lenguaje(basic,sencillo). Esto es verdadero porque existe como un hecho en la base de datos. Pasando al cut, éste triunfa automáticamente, pero luego fail hace que fracase toda la cláusula entera.

Sin el cut, el PROLOG habría retrocedido e intentado hallar otra forma de satisfacer Cláusula. No obstante, el cut se levanta como una pared infranqueable en su camino y, dado que no existe ninguna otra forma de satisfacer el objetivo, todo debe fracasar. Asimismo, debido al cut, el PROLOG no buscará cláusulas alternativas, de modo que faltante(Cláusula) acaba siendo falso (porque Cláusula era verdadera) y nosotros no desperdiciamos esfuerzos buscando alternativas.

Este ejemplo en particular ilustra un aspecto muy importante del PROLOG: puede tratar a su propio programa como un dato. Ello hace de este lenguaje una opción ideal para la programación de inteligencia artificial. Éste, junto con otras técnicas de programación avanzada del PROLOG, serán los temas del próximo capítulo.



Kevin Jones

De un lugar a otro
En el último capítulo del PROLOG crearemos un programa para hacer que un robot se desplace por la habitación ilustrada arriba. Sus posiciones se almacenan en el formato lugar(puerta)etc., y el robot se desplaza utilizando ir(Lugar1,Lugar2). Esto supone, en primer lugar, demostrar el subobjetivo lugar(Lugar1), es decir, establecer que el Lugar1 existe y que se puede efectuar un movimiento desde él. ¿Qué otros subobjetivos será necesario demostrar al objeto de planificar un programa de la ruta?

Complementos al PROLOG

El dialecto de PROLOG utilizado a lo largo de esta serie es el denominado DEC-10 PROLOG (en alusión al ordenador de Digital Equipment Corporation en el cual se implementó por primera vez), y está ampliamente reconocido como el estándar. Lamentablemente, la implementación del PROLOG más popular para microordenadores (MICRO-PROLOG) ¡es una de las pocas versiones que difieren de ésta! La otra implementación principal para micros (PROLOG-1) se acerca mucho al estándar DEC-10 (así como su sucesora, PROLOG-2). Los programas en MICRO-PROLOG se escriben enteramente como listas, de forma muy similar a la empleada en LISP. Sin embargo, al programador se le evita el esfuerzo de utilizar esta difícil notación mediante la ejecución de un programa pantalla denominado SIMPLE, que está escrito en MICRO-PROLOG y proporciona una sintaxis alternativa. Existen muchas pequeñas diferencias entre el MICRO-PROLOG y la versión estándar, que hacen que los dos dialectos ofrezcan un aspecto bastante diferente. El MICRO-PROLOG coloca al predicado como el primer elemento de una lista. Por ejemplo:

```
casa_de(marte,marciano).
```

aparecería como:

```
(casa_de Marte Marciano)
```

MICRO-PROLOG también elimina todos los conectores, tales como ':–' y comas, de modo que una cláusula como:

```
casa_de(Planeta,Criatura):–
nacido(Criatura,Ciudad),
en(Ciudad,Planeta).
```

tendría el siguiente aspecto:

```
((casa_de x y)(nacido y z)(en z x))
```

que se puede leer como una lista de tres sublistas, siendo la primera el encabezamiento de la cláusula. El programa *Simple* hace que esta sintaxis sea fácil de comprender y, con su notación de infijos para los predicados, la cláusula podría tener este aspecto:

```
x casa_de y si y nacido z y z en x
```

que sería aún más fácil de comprender si no fuera por el hecho de que el MICRO-PROLOG sólo permite x,y,z,X,Y o Z (o una de éstas seguida por un dígito) como nombres de variables. Entre las otras diferencias se incluye el empleo, por parte del MICRO-PROLOG, de paréntesis en vez de corchetes para las listas; la separación de argumentos y elementos de la lista con espacios y no con comas; la utilización de un '/' en lugar de un '!' para cut; y dar los predicados incorporados en letras mayúsculas. Asimismo, los predicados incorporados poseen nombres no estándares y, en algunos casos, no hacen exactamente las mismas cosas que sus equivalentes estandarizados; pero esto ya es bastante común entre los dialectos de lenguajes de programación. El MICRO-PROLOG sí posee una útil característica de la cual carece el DEC-10 y PROLOG: nos permite crear módulos de programas. Los módulos permiten escribir programas en unidades funcionales y luego combinarlas, sin preocuparse por conflictos entre los nombres de los predicados. Ello le confiere al PROLOG una cierta dosis de estructuración, pero, así y todo, el MICRO-PROLOG aún dista mucho de ser un lenguaje estructurado

Primero de su clase

Con una inmejorable relación calidad/precio, el RM Nimbus irrumpe confiadamente en el mercado de gestión

A la conquista de un nuevo territorio
El RM Nimbus representa en cierto modo una ruptura para sus fabricantes, Research Machines. Basado en el procesador Intel 80186, capaz de direccionar hasta un megabyte de memoria, la máquina sugiere que la empresa está yendo más allá de su base educativa, establecida ya sólidamente, con las miras puestas en un campo sumamente lucrativo: el mercado de gestión

Research Machines Ltd es un nombre no muy conocido en círculos ajenos al educativo. En los últimos años la empresa se ha establecido como proveedora de microordenadores de ocho bits de gran calidad en escuelas y colegios. El ordenador ampliable 380Z y la máquina en red 480Z causaron un gran impacto desde el momento en que se llevó a cabo en Gran Bretaña el proyecto gubernativo "Los microordenadores en la educación", convirtiéndose en la única competencia seria para el BBC Micro de Acorn.

A pesar del hecho de que ordenadores como el 380Z se consideraron revolucionarios en el momento de su lanzamiento, ahora existe una demanda de mayor potencia de proceso, y esto sólo lo pueden proporcionar los procesadores de 16 bits. En respuesta a esta demanda, Research Machines ha lanzado un nuevo micro de 16 bits bajo el nombre de Nimbus.



Chris Stevens

Obviamente, desde que apareciera el 380Z la industria ha experimentado un considerable desarrollo. El extremo superior del mercado en la actualidad está dominado por el IBM PC y toda una multitud de "compatibles", la mayoría de los cuales se basan en el procesador 8088. Aunque el 8088 es más rápido que el Z80, ya está comenzando a parecer algo anticuado. En favor de la empresa se debe reconocer que, al desarrollar el Nimbus, no ha caído en la tentación de producir simplemente otro clono del IBM. En cambio, ha optado por basar la máquina en el chip 80186, mucho más veloz, un auténtico procesador de 16 bits capaz de ejecutar hasta un millón de instrucciones por segundo. El resultado es un ordenador que deja muy rezagada a la mayor parte de la competencia.

Aunque el Nimbus no es compatible con el IBM, el 80186 es un pariente cercano del 8088 y opera bajo el sistema operativo MS-DOS estándar. Asimismo, el teclado también se puede configurar al formato IBM, así como el sistema más familiar de Research Machines utilizado para el 380/480Z. Así, el teclado está equipado con 10 teclas de función programable, un teclado numérico y una letra para ALTernar el juego de caracteres. El tacto de las teclas es excelente en comparación con cualquier otro del mercado.

La versión que presentamos aquí es el Nimbus PC2. Equipado con unidades de disco gemelas de 3 ½ pulgadas formato Sony, existe también una versión con unidad de disco individual, denominada PC1. Cada uno de estos discos puede almacenar hasta 720 K de información y están siendo adoptados por una cantidad creciente de productos de "tecnología punta".

La decisión de incorporar estas unidades, así como otras varias características avanzadas, probablemente haya sido lo que disuadió a RM de producir una máquina compatible con el IBM, en favor de un ordenador más en consonancia con la "tecnología punta". La máquina IBM utiliza el antiguo formato de 5 ¼ pulgadas, que es más delicado que su equivalente Sony y posee una menor capacidad de almacenamiento. Hubiera parecido extraño que RM instalara unidades de disco que están quedando progresivamente anticuadas y que hubieran entorpecido el rendimiento de su máquina, sobre todo si se tienen en cuenta sus otras características avanzadas.

Esto plantea el problema de la compatibilidad de software entre los anteriores ordenadores de Research Machines, que utilizaban discos de 5 ¼ pulgadas, y el Nimbus. La empresa lo ha solventado proporcionando facilidades para permitir la adición de otras unidades de disco, incluyendo, por supuesto, una de 5 ¼ pulgadas. Esto significa que el Nimbus puede leer unidades de 5 ¼ pulgadas en formatos PC-DOS, MS-DOS o CP/M, característica que representa un sólido argumento de venta para todos aquellos usuarios de ordenadores de forma-

tos diferentes que preferirían no renunciar a todo su software actual.

Debajo de las unidades hay dos conectores para cartuchos de ROM. Éstos son compatibles con la puerta para cartuchos instalada en el 480Z, aunque aún es escaso el software disponible basado en ROM. A la izquierda hay un conector que permite la instalación de un *dongle*. Éste es un dispositivo para proteger el software que impide que los usuarios ejecuten un programa sin uno, aun cuando posean una copia del software.

El software que se proporciona con el Nimbus incluye un disco de sistema MS-DOS, el *Wordplan* y *Multiplan* de Microsoft, así como el LOGO y el BASIC propios de RM. Cuando se carga el software, uno comienza a apreciar la excelente máquina que es el Nimbus en realidad. En especial, son sorprendentes las facilidades para gráficos proporcionadas desde BASIC.

El BASIC RM

El BASIC que viene con el Nimbus es un desarrollo de la Version 5 de RM. Teniendo en cuenta que la base fundamental de RM es la del mercado educativo, no es sorprendente que el lenguaje posea varias instrucciones diseñadas para fomentar la programación estructurada, si bien no tantas como la ulterior Version 6 BASIC. El BASIC RM del Nimbus contiene estructuras de instrucción tales como REPEAT..UNTIL, PROC..ENDPROC y GLOBAL, pero carece del posterior DO..WHILE.

Por otra parte, Research Machines ha añadido algunas instrucciones útiles diseñadas para serles de ayuda a los programadores de juegos. Las instrucciones JOYX, JOYY y MOUSE pueden utilizar la palanca de mando y el ratón opcionales; la instrucción BUTTON inicia los efectos producidos por estos dispositivos.

Los gráficos, el sonido y otros atributos se seleccionan desde BASIC utilizando la instrucción SET seguida de los parámetros que se requieran. Mediante el uso de esta instrucción se pueden seleccionar colores de primer plano y fondo (definidos por las instrucciones PAPER, BRUSH y PEN), dibujar líneas y círculos y seleccionar fuentes, direcciones y tamaño de los caracteres. También es posible elegir lo que el manual describe como "estilos" y "patrones vacilantes", produciendo el efecto de un color que es una combinación de los puntos comprendidos en el área.

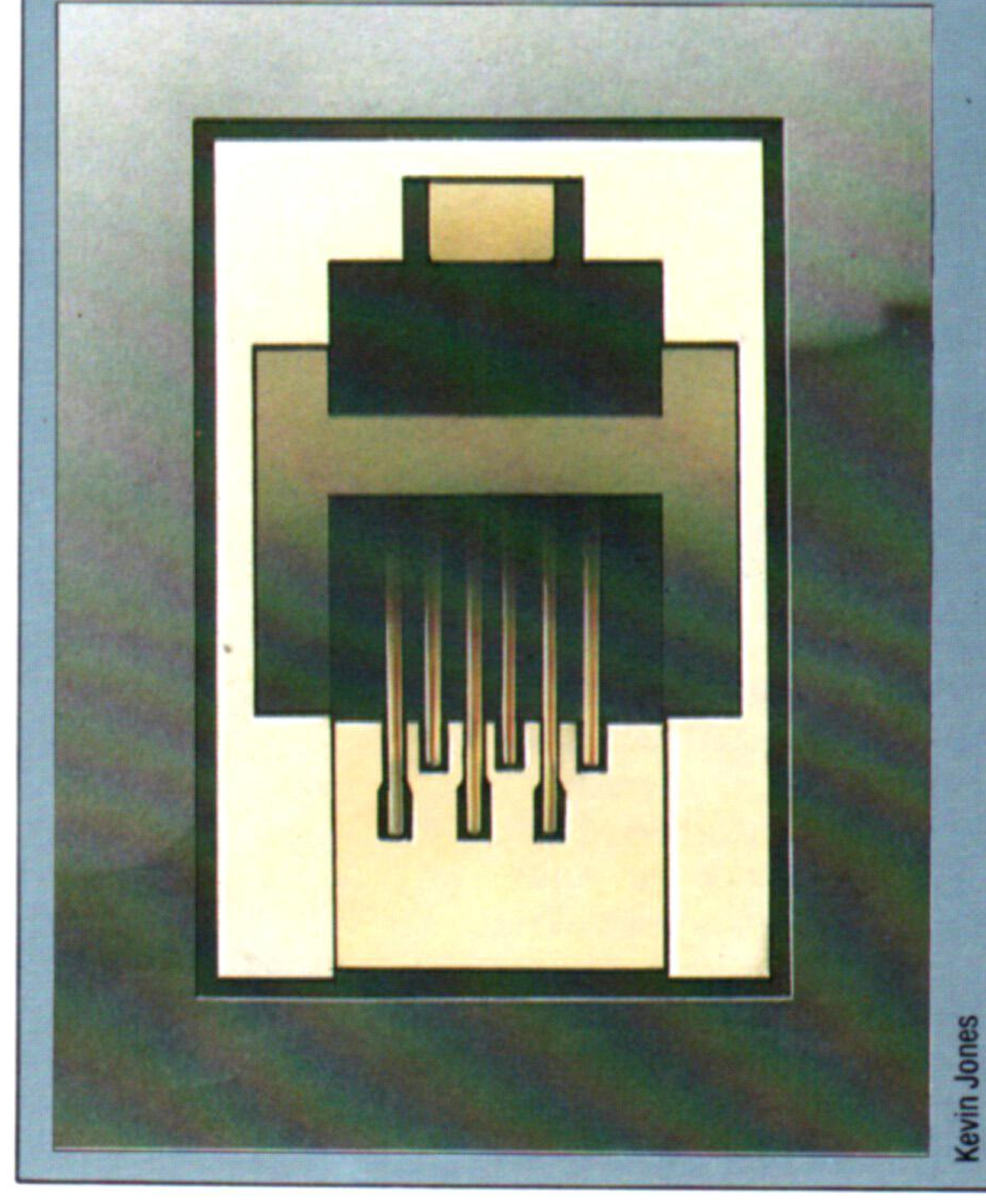
Aparte de la amplia gama de efectos que se pueden producir en el Nimbus, es notable la velocidad a la cual se dibujan los gráficos en la pantalla desde BASIC, comparable a muchos ordenadores de 16 bits programados en código máquina.

Los discos se organizan en directorios y subdirectorios. Esto significa, por ejemplo, que para poder entrar BASIC, usted debe entrar al directorio de BASIC desde el directorio principal. Esto se realiza desde MS-DOS y el propio BASIC mediante la instrucción CHange DIRectory (cambiar directorio), CD, para más brevedad. Siguiendo esta instrucción, el usuario debe especificar los nombres de "senderos" que se requieren con el fin de obtener el directorio en cuestión. Esto puede sonar un poco complicado, pero permite (y también estimula a ello) organizar los directorios de una forma sistemática. Desde el sistema operativo es posible la creación de nuevos senderos y directorios, además del establecimiento de nuevos senderos entre directorios existentes.

Siendo fiel a su filosofía de diseño para el Nimbus, Research Machines ha optado por la tecnología del "estado del arte" en vez de las puertas para periféricos estándares. El Nimbus posee tres líneas de salida de potencia separadas desde la unidad principal del ordenador. En la esquina superior izquierda de la máquina se halla una salida de 2A desde el transformador principal de potencia, y a lo largo del panel posterior hay dos salidas más pequeñas, de 12 V y 5 V.

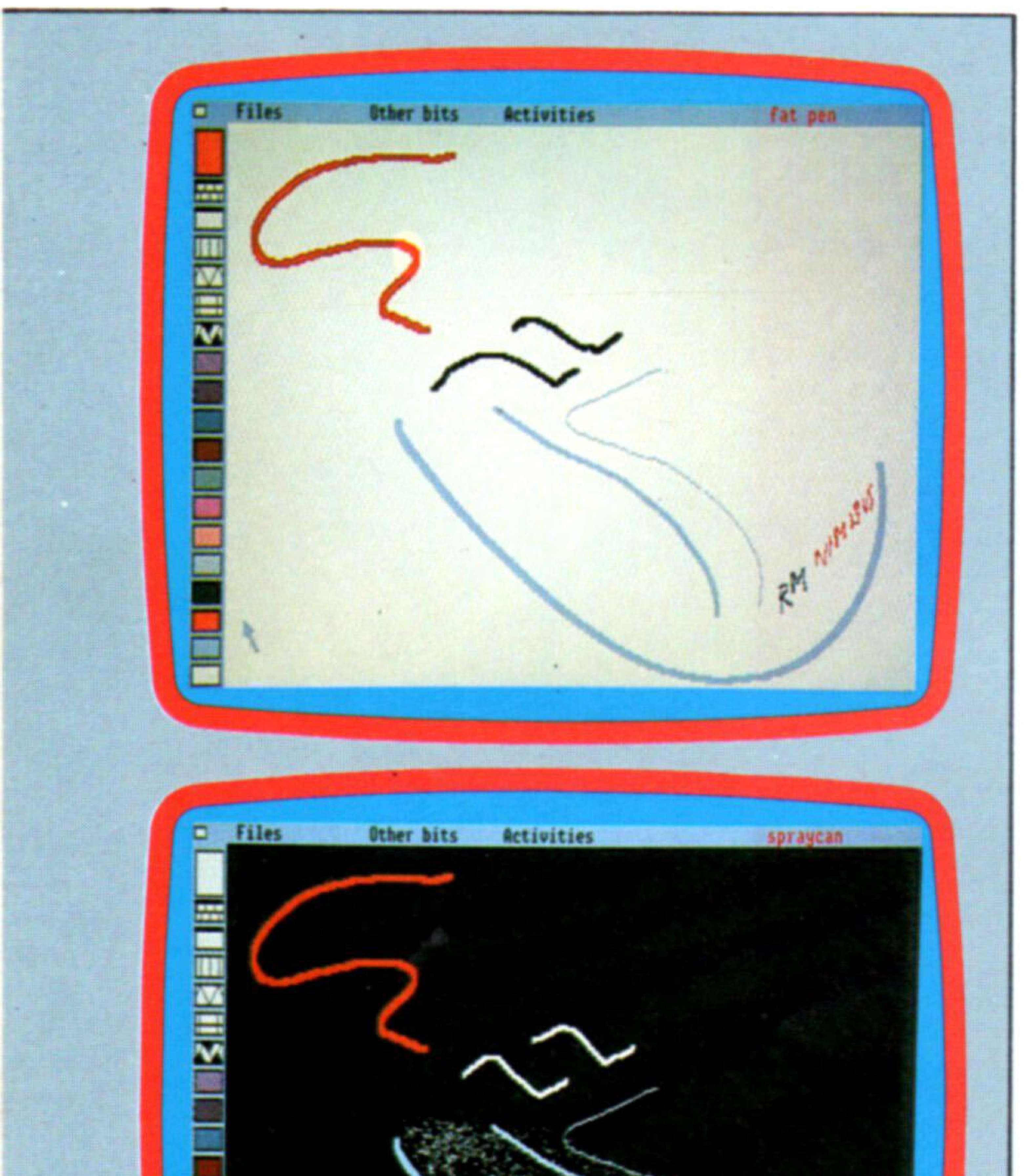
Entre las otras interfaces que se proporcionan hay una puerta Piconet, que se instala como un enchufe telefónico BT estándar. El Piconet permite





Kevin Jones

Ampliación potencial
El RM Nimbus está bien provisto de interfaces. El ordenador posee dos conectores para pantalla, una puerta para ratón/palanca de mando, y una interface Piconet, puerta para impresora y no menos que tres líneas de potencia auxiliares. Las importantes puertas de E/S Piconet y para impresora utilizan el estándar de enchufe BT introducido recientemente, que muchos usuarios con equipos más antiguos no podrán conectar de forma directa. El enchufe de carrillón BT (que vemos a la izquierda) es una interface en serie capaz de manipular señales tanto de entrada como de salida. Hay seis líneas disponibles, que se pueden disponer en numerosas configuraciones para transmisión, interconexión y otras aplicaciones



MacNimbus
Entre los programas de demostración que se proporcionan con el Nimbus hay un paquete de gráficos activado por ratón. Al igual que muchos programas de dibujo por ratón, la pantalla guarda un notable parecido con la del Apple Macintosh. Sin embargo, a diferencia de éste, en la pantalla se pueden visualizar hasta 16 colores distintos al mismo tiempo

Ian McKinnell

Chip de E/S
La gestión de entrada/salida se controla desde este procesador

Chip Z80A
Al objeto de poder leer y procesar discos RM formateados para usar con ordenadores 380Z/480Z, RM ha incluido como segundo procesador el familiar Z80A

Placa de RAM
Esta versión particular del ordenador posee RAM extra proporcionada por esta placa accesoria

Chips de RAM
En su versión estándar, el Nimbus viene equipado con 128 K de RAM

Interface Soft Key
Conocida corrientemente como puerta "dongle", esta interface asegura que, cuando no está instalado un dongle, sea imposible copiar programas sin autorización

Control de disco flexible
Esta placa, que está instalada en una de las ranuras de ampliación, controla las unidades de disco

Unidades de disco
El Nimbus PC2 viene equipado con un par de unidades de disco formato Sony

gobernar desde una única puerta hasta 30 periféricos separados, cada uno de ellos conectado a través de un módulo externo. La conexión en interface se realiza en serie y está construida de acuerdo al estándar RS422. Es esta nueva conexión de alta velocidad la que permite al Nimbus controlar los 30 dispositivos.

Lamentablemente, en muchos casos el RS422 no es directamente compatible con el anterior estándar RS232, de modo que si usted desea utilizar estos dispositivos habrá de obtener interfaces adecuadas.

También se proporciona una facilidad de red que permite conectar el Nimbus a una red compuesta por otros ordenadores de Research Machines a través del conector coaxial.

Interface para impresora

La interface para impresora es una inclusión curiosa. Se ha instalado como un enchufe telefónico BT idéntico a la puerta Piconet, y no responde exactamente al estándar. Además, la puerta únicamente es en serie y está diseñada de acuerdo al estándar RS422.

El resto de las interfaces que se proporcionan ya son más familiares. Hay un conector D de nueve patillas que facilita la conexión al ordenador de una





palanca de mando o, más probablemente, un ratón, al objeto de ejecutar aplicaciones tipo Macintosh (los programas de demostración contienen un programa de dibujo que, aparte de los gráficos en color, guarda un notable parecido con la pantalla Macintosh, si bien carece de muchas de las facilidades de *MacPaint*).

Se proporcionan dos conectores para pantalla, uno de los cuales, para monitores RGB, ejecuta la pantalla en color Cub que RM ha diseñado especialmente. El otro es un conector de video compuesto destinado a pantallas monocromáticas. Por último, hay un enchufe DIN para el teclado.

En el extremo izquierdo (mirando hacia la parte posterior de la máquina) hay cuatro puertas de ampliación en las que el usuario puede agregar placas extras. Éstas incluirían bancos adicionales de RAM, interfaces digital/analógico, etc. A diferencia de otros muchos fabricantes que insisten en que esta clase de trabajo sólo lo han de realizar los representantes autorizados, en su guía para el usuario Research Machines ha ofrecido instrucciones completas para que usted instale sus propias placas.

Documentación completa

El manual que se entrega con el Nimbus responde muy bien al elevado estándar que la gente ha llegado a esperar de RM. La información que contiene incluye una guía para principiantes sobre el MS-DOS y los discos, junto con un estudio detallado de las puertas.

Lo único que se echa en falta es un análisis del chip procesador 80816 que, aunque probablemente no fuera de gran utilidad para la inmensa mayoría de usuarios, sí serviría a los programadores que pretendieran desarrollar su propio software en la máquina.

Aún es demasiado pronto para decir a quiénes les resultará más atrayente el Nimbus. Decididamente, es una máquina de una gran calidad que, de entrada, vale casi la mitad que su único rival serio, el IBM PC/AT. No cabe duda de que el ordenador tendrá una buena acogida, pero el volumen de ventas dependerá de la estrategia de comercialización de RM.

Las universidades y otros grandes establecimientos educativos que ya tienen establecidas buenas relaciones con RM son potenciales adquirentes de la máquina, y no se preocuparán demasiado por el apoyo de software: con toda probabilidad, desearán escribir el suyo propio y podrán transferir el software RM que ya posean. Las velocidades y las facilidades para una programación sencilla representarán, obviamente, convincentes argumentos de venta, así como la cantidad de periféricos que se pueden instalar.

Sin embargo, para que el Nimbus satisfaga su potencial en el mercado de gestión (potencial que, evidentemente, está al alcance del ordenador), Research Machines habrá de convencer a los usuarios de gestión de que pondrá a su disposición el software adecuado. RM habrá, asimismo, de adaptar las interfaces al estándar Centronics y RS232 que se utiliza en la mayoría de las oficinas. Si Research Machines consigue introducirse en el sector de gestión, el Nimbus podría convertirse en un importante desafío para los actuales líderes del mercado, ACT/Apricot e IBM.

Chris Stevens

RM NIMBUS

DIMENSIONES
400×350×110 mm

CPU
Intel 80186 operando a 8 MHz

PANTALLA
Máximo de 80×25 caracteres de texto, 640×250 pixels en modalidad a cuatro colores, 320×125 en modalidad a ocho colores

INTERFACES
Enchufe para red, puerta Piconet, puerta para impresora, salidas de potencia de 12 V, 5 V y 2A, conectores para monitor compuesto y RGB

LENGUAJES DISPONIBLES
BASIC RM, LOGO RM y PASCAL RM

TECLADO
83 teclas incluyendo teclado numérico y 10 teclas de función programable

DOCUMENTACION
Los manuales son exhaustivos y contienen la mayor parte de la información que puede necesitar el usuario. Asimismo, parecen ser menos densos y más fáciles de leer que los de la documentación anterior

VENTAJAS
El Nimbus tiene muy pocos competidores a su mismo precio. La tecnología avanzada, junto con su sencillez de uso, excelentes gráficos y buenos manuales, harán que la máquina se convierta en un producto de éxito

DESVENTAJAS
Al menos a corto plazo, quizá la tecnología sea demasiado avanzada; el resto de la industria no está aún a la altura del Nimbus y, por tanto, tal vez haya escasez de ciertas interfaces

Muy corriente

Veamos ahora cómo fluyen los datos a través de los canales hasta la impresora o la pantalla en un Spectrum

Decíamos que siempre que se enciende o restaura el Spectrum, unas determinadas corrientes se asocian con determinados canales en el proceso de inicialización. Estas asociaciones son las que resumimos en el siguiente cuadro:

Canal	Corriente	Descripción del canal
K	0	Teclado y parte inferior de la pantalla
K	1	
S	2	Parte principal de la pantalla
P	3	Impresora ZX

Sin embargo existen otras corrientes a nuestra disposición no empleadas por el Spectrum normal. Se observará que la siguiente descripción corresponde a un Spectrum sin Interface 1; todo se altera si empleamos redes de área local, microdrive o interface serial.

Estos dispositivos serán analizados más adelante. Las otras corrientes están numeradas del 4 al 15, y pueden también asociarse con cualquier canal de E/S que se necesite intercomunicar.

El OS del Spectrum registra cuál de las corrientes se asocia a cada canal en uso. Esta información se guarda en una tabla de 38 bytes llamadas STRMS (de *streams*: corrientes), que puede encontrarse en el área reservada para las variables de sistema con dirección 23568. Hay seis bytes al comienzo de la tabla empleados por el OS del Spectrum para sus propios fines. Según éstos, cada una de las 16 corrientes está representada por una entrada de dos bytes. La disposición de la tabla es la que muestra el dibujo.

El siguiente fragmento en BASIC imprimirá el contenido de este área de la memoria para que usted pueda conocerlo:

```
10 FOR I=23568 TO 23604 STEP 2
20 PRINT PEEK I+256*PEEK(I+1)
30 NEXT I
```

Nos dará todas las entradas de dos bytes. Las tres primeras son los detalles de las corrientes internas del Spectrum, siguiendo después con las entradas de las corrientes de la 0 a la 15. Los valores en esta última parte de la tabla actúan como punteros de otras entradas en una tabla distinta, situada en cualquier parte de la memoria y a la cual apunta una variable de sistema llamada CHANS. Esta variable se encuentra en la dirección 23631, y la dirección de la tabla de CHANS puede hallarse así:

```
PRINT PEEK 23631+256*PEEK 23632
```

Esta variable de sistema *nunca* será tratada con POKE. La expresión que acabamos de darle proporciona el inicio de la *tabla de registro de canales*, que contiene varios detalles informativos sobre cada canal.

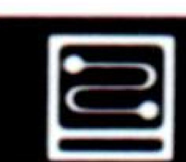
El valor leído de la tabla STRMS sirve de índice de esta tabla de registro de canales. Una entrada de la tabla STRMS da el (desplazamiento + 1) desde el comienzo de la tabla de registro de canales de la información sobre el canal que se está usando con una determinada corriente. Así la entrada de 1 tabla STRMS para corriente 3 (a la impresora) es 16. Si se resta uno de éste obtenemos un desplazamiento de 15. Por lo tanto los datos correspondientes al canal P se encontrarán en la dirección:

```
(PEEK 23631+256*PEEK 23632)+15
```

(siempre suponiendo que se acaba de encender el ordenador y no se han alterado las asociaciones corrientes-canales después de la inicialización).

Si los datos que están en una entrada de la tabla STRMS son cero, esto indica que la corriente todavía no ha sido asociada con ningún canal. Éste es el caso, en un Spectrum no ampliado, de las corrientes que van de la 4 a la 15 después de la inicialización.

Cada entrada en la CRT es de la forma:



byte *lo*	Dirección de la rutina de salida para el canal
byte *hi*	
byte *lo*	Dirección de la rutina de entrada para el canal
byte *hi*	
n	Byte único que representa la letra código del canal en ASCII (así, K, S y P)

Una entrada de la tabla STRMS apunta al segundo byte de cada entrada de la tabla de registro de canales (CRT), por eso hay que restar uno antes de su empleo. Esta tabla está en el corazón mismo del sistema de E/S del Spectrum. Si un canal no puede llevar a cabo una operación de E/S, la entrada de la dirección adecuada en la tabla CRT apuntará a una rutina en la ROM que generará el mensaje de error Invalid I/O Device (dispositivo de E/S ilegal). Así, para un canal de sólo salida (como es el canal S) la dirección de la rutina de entrada apuntará a esta rutina de error. En caso contrario, la rutina adecuada de la ROM para salidas o entradas serían las apuntadas por las entradas de la tabla.

Examinemos la entrada del canal P en la tabla. Recuérdese que en circunstancias normales es un canal de sólo salidas, que envía datos a la impresora ZX. Los datos para este canal en la entrada de la CRT son:

(CHANS)+15⇒	8'F4	Apunta a la rutina GP Print Out en la dirección 09F4 en la ROM
	8'09	
	8'C4	Apunta a la rutina generadora de error en la ROM, dirección 15C4
	8'15	
	8'50	Código ASCII de "P"

La entrada para el canal S se halla en (CHANS)+5, y los datos del canal K están en (CHANS)+0. (CHANS) es la dirección retenida en la variable de sistema CHANS.

Con las instrucciones OPEN# y CLOSE# podemos asociar corrientes no empleadas a uno de los canales que hasta ahora hemos examinado. Pero estas dos instrucciones realmente se utilizan para asociar las corrientes a otros canales en el sistema ampliado, tales como los que ofrece la Interface 1. Incluso de esta manera examinaremos brevemente el empleo de estas instrucciones sin Interface 1. Por ejemplo,

```
OPEN #4,"S"
```

asociará la corriente cuatro al canal S. Si se realiza esta instrucción y después se examina la tabla STRMS, se verá que se ha hecho una entrada en la posición correspondiente a la corriente 4. En la sentencia podemos hacer uso de variables.

```
10 LET n=4
20 LET c$="S"
30 OPEN #n,c$
```

hará exactamente lo mismo que la sentencia anterior. Cuando se abre un canal de esta forma, las instrucciones PRINT# y LIST# le permitirán enviar datos al canal a través de la nueva corriente. Así podemos añadir:

```
40 PRINT #4; "Hola"
```

que se imprimirá en la pantalla. LIST#4 dará un listado del programa que está en pantalla según sabemos. Si una corriente se ha asociado con un canal tanto de entrada como de salida, se puede usar también la instrucción INPUT#. Estas sentencias PRINT e INPUT ampliadas sólo tienen utilidad cuando se puede disponer de los canales que proporciona la Interface 1, como veremos.

Cuando se finaliza con una determinada combinación de corriente y canal, y se desea disolver el emparejamiento, usaremos CLOSE#. Así, en nuestro ejemplo, CLOSE#4 independizará la corriente 4 del canal S. Llegados aquí, todo intento de enviar datos a esta corriente "cerrada" generará un mensaje de error.

Las instrucciones PRINT#,LIST#,INPUT# e INKEY# pueden usarse también con las corrientes de la 0 a la 3.

Estas corrientes están siempre abiertas, como puede comprobar ejecutando lo siguiente:

```
10 CLOSE#2: REM debería cerrar el canal S
20 PRINT#2; "Hola"
```

No aparecerá mensaje alguno de error (el OS se encarga de abrir el canal de nuevo antes de pasarle información). Consideremos ahora la instrucción LIST#.

```
LIST#3
```

equivale, en circunstancias normales, a LLIST: el programa se lista en la corriente 3, la cual, como acabamos de ver, se asocia con el canal P. Las instrucciones LIST#0 y PRINT#0 son bastante interesantes ya que éstas llegan a imprimir en la parte inferior de la pantalla (que normalmente cae fuera del dominio de las sentencias PRINT). Llegados a este punto, adviértase que cualquier texto que se halle impreso en esta parte de la pantalla será sobreescrito por medio de una indicación o mensaje de error del sistema operativo. En circunstancias normales la instrucción PRINT#3 equivale a LPRINT.

Por último, se puede modificar el comportamiento de un canal en el Spectrum para que realice algo diferente de aquello para lo cual fue inicializado. Por ejemplo, el canal P es modificado a menudo para permitir que una impresora Centronics sea controlada por el Spectrum y una interface apropiada. Si alteramos la entrada de la rutina de salida en la entrada adecuada de la CRT, las instrucciones LPRINT y LLIST enviarán datos a una rutina que maneja la impresora Centronics. Naturalmente, no resulta preciso que se trate de una impresora; puede ser una puerta de salida o cualquier dispositivo de salida.



Programa "Imprimir en alt-res"

Este programa intercepta los datos que se envían por el canal P y los desvía a una rutina que permite al usuario la facilidad de imprimir en alta resolución. El listado puede ser entrado mediante un ensamblador corriente (el CHAMP, DevPac o Picturesque, p. ej.). Si usted carece de un ensamblador tendrá que entrarlo en codificación hexadecimal, empleando un monitor o bien mediante el Cargador en Hexa que aquí ofrecemos. Una vez digitado este cargador y ejecutado, se le pedirá que digite la dirección de inicio (que es 65068). Se espera ahora que digite el código hexa, que puede ser entrado sólo mediante dos dígitos a la vez. Por ejemplo, la línea del código:

2130FE

habrá de ser digitada de esta manera:

21 (ENTER)
30 (ENTER)
FE (ENTER)

Entrado el código, falta tan sólo pulsar S para dar por finalizado el programa. Es el momento de la suma de control (que es 29155), que le indicará si cometió algún error. La nueva instrucción que proporciona el programa tiene este formato:

LPRINT AT X,Y;"MENSAJE"

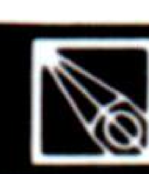
donde X debe valer entre 0 y 168, e Y entre 0 y 248. En vez de "MENSAJE" puede estar una variable en serie o un número. Cuando la serie que se imprime con la nueva instrucción sobrepasa un borde de la pantalla, vuelve a empezar por el comienzo de la misma línea, pero no pasa a una línea más abajo. Las coordenadas X e Y funcionan de idéntico modo que con la instrucción PLOT del Spectrum

Cargador hexa

```
 10 CLEAR 63999
 15 POKE 23658,8
 17 DEF FN H(H$)=16*(CODE
    (H$)-48-7*(H$(1)>"9"))+(CODE
    (H$(2))-48-7*(H$(2)>"9"))
 20 INPUT "Dirección inicio"; a
 25 LET Z=A
 30 INPUT (a), LINE a$
 35 IF A$="S" THEN GO TO 90
 40 PRINT a,a$: POKE 23692,255
 50 FOR Z=1 TO LEN A$/2
 60 POKE a,FN H(a$(((z*2)-1) TO)): LET a=a+1
 70 NEXT z
 80 GO TO 30
 90 INPUT "CHECKSUM:";D
100 LET C=0: FOR B=Z TO A-1: LET C=C+PEEK B:
    NEXT B
110 IF C=D THEN PRINT "Codificación perfecta"
120 PRINT "Vaya, la codificación no ha sido correcta"
```

Listado assembly

```
                  ORG    65068
         PIXAD    EQU    22AAH
         UDG      EQU    23675
         CHARS    EQU    23606
         ;               "Hace que los datos en CHAN apunten a nuestra rutina"
2A4F5C   ENABL    LD     HL,(23631)
010F00            LD     BC,15
09                ADD    HL,BC
013AFE            LD     BC,DO-IT
71                LD     (HL),C
23                INC    HL
70                LD     (HL),B
C9                RET
         ;               "Guarda regs. y llama a nueva rutina impresora"
E5       DO-IT    PUSH   HL
C5                PUSH   BC
D5                PUSH   DE
F5                PUSH   AF
CD46FE            CALL   DOIT1
F1                POP    AF
D1                POP    DE
C1                POP    BC
E1                POP    HL
C9                RET
         ;               "Comprueba si a contiene el cód. de control de AT"
F5       DOIT1    PUSH   AF
3A1DFF            LD     A,(ATFLG)
FE00              CP     0
200B              JR     NZ,GETXP
F1                POP    AF
FE16     ATCHQ    CP     22
201D              JR     NZ,CRCHQ
3EFF              LD     A,255
321DFF            LD     (ATFLG),A
C9.               RET
FEFE     GETXP    CP     254
2809              JR     Z,GETYP
F1                POP    AF
3216FF            LD     (XPOSI),A
211DFF            LD     HL,ATFLG
35                DEC    (HL)
C9                RET
F1       GETYP    POP    AF
3217FF            LD     (YPOSI),A
3E00              LD     A,0
321DFF            LD     (ATFLG),A
C9                RET
         ;               "Comprueba el dato de fin de impresión Z(que aquí es el 13)"
FE0D     CRCHQ    CP     13
2001              JR     NZ,VCHRQ
C9       SKIPC    RET
         ;               "Si el código en a está entre 32 y 128 o si"
         ;               "es un cód. UDG entonces halla su dato en la mem."
FE20     VCHRQ    CP     32
380C              JR     C,PRNT?
FE80              CP     128
381C              JR     C,FCHR
FE90     UDGCQ    CP     144
3804              JR     C,PRNT?
FEA5              CP     165
3804              JR     C,FUDGC
         ;               "Para todos los demás códigos imprimirá un    ?"
3E3F     PRNT?    LD     A,63
1810              JR     FCHR
         ;               "Halla el dato de UDG o de car. en la mem."
D690     FUDGC    SUB    144
210000            LD     HL,0
6F                LD     L,A
29                ADD    HL,HL
29                ADD    HL,HL
29                ADD    HL,HL
EB                EX     DE,HL
2A7B5C            LD     HL,(UDG)
19                ADD    HL,DE
180C              JR     PRNIT
210000   FCHR     LD     HL,0
6F                LD     L,A
29                ADD    HL,HL
29                ADD    HL,HL
29                ADD    HL,HL
EB                EX     DE,HL
2A365C            LD     HL,(CHARS)
19                ADD    HL,DE
010700   PRNIT    LD     BC,7
09                ADD    HL,BC
221BFF            LD     (CHRAD),HL
         ;               "Comprueba si las posiciones X e Y son correctas"
3A16FF            LD     A,(XPOSI)
FEF9              CP     249
D231FF            JP     NC,ERRB
3A17FF            LD     A,(YPOSI)
FEA9              CP     169
3078              JR     NC,ERR5
ED4B16FF          LD     BC,(XPOSI)
CDAA22            CALL   PIXAD
321AFF            LD     (PIXPO),A
2218FF            LD     (DFADD),HL
         ;               "Toma el primer conjunto de pixels o car. impreso"
0608              LD     B,8
C5       PRNLP    PUSH   BC
2A1BFF            LD     HL,(CHRAD)
7E                LD     A,(HL)
2B                DEC    HL
221BFF            LD     (CHRAD),HL
6F                LD     L,A
         ;               "Si el conj. de pixels primero no ha de moverse"
         ;               "dentro del byte de visual., éste salta adelante"
3A1AFF            LD     A,(PIXPO)
FE00              CP     0
CAE6FE            JP     Z,PUTIT
         ;               "Mueve el conj. de pixels en los bytes de visual."
         ;               "hacia la posición correcta del pixel"
47                LD     B,A
2600              LD     H,0
CB3D     ROTLP    SRL    L
CB1C              RR     H
A7                AND    A
10F9              DJNZ   ROTLP
         ;               "Lo pone en la posición derecha de la pantalla"
ED5B18FF PUTIT    LD     DE,(DFADD)
1A                LD     A,(DE)
AD                XOR    L
12                LD     (DE),A
3A1AFF            LD     A,(PIXPO)
FE00              CP     0
CAFAFE            JP     Z,PST
13                INC    DE
1A                LD     A,(DE)
AC                XOR    H
12                LD     (DE),A
1B                DEC    DE
2A18FF   PST      LD     HL,(DFADD)
CD1EFF            CALL   ULINE
2218FF            LD     (DFADD),HL
C1                POP    BC
10C4              DJNZ   PRNLP
         ;               "Añade 8 a la posic. X para que el siguiente car."
         ;               "se imprima en espacio borrado"
3A16FF            LD     A,(XPOSI)
C608              ADD    B
         ;               "Si se llega al fin de línea vuelve al"
         ;               "inicio PERO NO baja una línea"
FEF9              CP     249
DA12FF            JP     C,STAIT
3E00              LD     A,0
3216FF   STAIT    LD     (XPOSI),A
C9                RET
00       XPOSI    DEFB   0
00       YPOSI    DEFB   0
0000     DFADD    DEFW   0
00       PIXPO    DEFB   0
0000     CHRAD    DEFW   0
00       ATFLG    DEFB   0
         ;               "Mueve la direc. en HL un pixel arriba de la pant."
F5       ULINE    PUSH   AF
7C                LD     A,H
25                DEC    H
E607              AND    7
200A              JR     NZ,END
7D                LD     A,L
D620              SUB    32
6F                LD     L,A
3804              JR     C,END
7C                LD     A,H
C608              ADD    8
67                LD     H,A
F1       END      POP    AF
C9                RET
         ;               "Crea varios errores"
CF       ERRB     RST    8
0A                DEFB   10
CF       ERR5     RST    8
04                DEFB   4
         FINIS    END
```



Sistema sonoro

Iniciamos un nuevo proyecto: el diseño y la construcción de una interface MIDI para el BBC Micro y el Commodore 64

Las especificaciones de la MIDI (*Musical Instrument Digital Interface*: interface digital para instrumentos musicales) esbozan el hardware y el software necesarios para un sistema que permita que los instrumentos equipados con la interface se comuniquen entre sí. Puesto que la comunicación es de una naturaleza digital, en el sistema MIDI podemos también introducir un micro personal para controlar otros instrumentos o almacenar datos en su memoria, ya sea en cinta o en disco. Las especificaciones de hardware de la interface nos permiten construir una placa de circuitos conectable, fácil de construir y económica.

Una vez construida la interface, se puede programar al ordenador para que interactúe con otros dispositivos del sistema MIDI, escribiendo el software adecuado. Más adelante incluiremos estos listados.

Desde la introducción de las primeras especificaciones MIDI, en agosto de 1983, el mundo de la música electrónica se ha visto inmerso en un ambiente de confusión y malentendidos sobre lo que la interface puede y no puede hacer. Esto se debe a que a los músicos no les preocupa la naturaleza de la comunicación entre los elementos electrónicos, sino que la mayoría de ellos sólo se interesan (y con razón) en lo que un sistema determinado les puede ofrecer en términos de actuación musical.

A lo largo del curso explicaremos las especificaciones MIDI con una terminología que les resultará más familiar a los programadores que a los músicos, y les proporcionaremos a usted, de este modo, la misma información básica que utilizan los diseñadores de equipos que pretenden que sus productos sean compatibles con el estándar.

En primer lugar hemos de hacernos alguna idea sobre la naturaleza de los datos comunicados por la MIDI. La mejor manera de entenderlo es considerando las diversas partes de un sistema de música electrónica como los periféricos de un ordenador. Puede ser que la distinción entre estos componentes no se haga evidente de inmediato: por ejemplo, un típico sintetizador polifónico moderno en reali-

Hardware MIDI estándar

Por lo general la interface consta de dos secciones: una para recibir datos y otra para transmitirlos. Cada una posee sus propios conectores, etiquetados MIDI IN y MIDI OUT, respectivamente.

Puede haber un tercer conector, MIDI THRU, que envía una copia de los datos en MIDI IN. Esto es para facilitar la conexión de receptores múltiples en cadena sin la necesidad de múltiples MIDI OUT en el transmisor. Abajo vemos el diagrama de circuitos del hardware de la interface

Reproducido por cortesía del International MIDI Users Group (grupo internacional de usuarios de MIDI)

Notas:
1. El optoaislante que se indica es un Sharp PC-900 (puede utilizarse un HP 6N138 u otro optoaislante modificado)
2. Puertas "A": IC o transistor
3. Las resistencias son 5 %

Kevin Jones

dad está constituido por dos componentes separados, si bien por lo general se hallan contenidos en la misma caja. Los dos componentes son el teclado (el dispositivo de entrada) y el sistema de circuitos para la generación de sonido (el dispositivo de salida). En la modalidad de operación normal, el dispositivo de entrada está conectado directamente con el dispositivo de salida: la pulsación de una tecla produce de inmediato una salida de sonido.

Esta situación es similar a la de una sofisticada máquina de escribir electrónica, en la cual los dos componentes (el teclado y el mecanismo de impresión) se pueden conectar entre sí de modo que los datos de entrada se reproduzcan de forma inmediata. Sin embargo, es muy fácil reconocer el teclado y la impresora como los periféricos potenciales de un ordenador central, que leerá los datos de entrada y los *procesará* antes de enviarlos al dispositivo de salida. Esto, por supuesto, ya nos resulta familiar: se trata de un procesador de textos.

En el caso del sintetizador es importante destacar que la interface (interna) entre el teclado y los generadores de sonido es puramente digital. El teclado actúa simplemente como un controlador del sintetizador, de la misma forma en que el teclado de la máquina de escribir actúa como un controlador para el mecanismo de impresión. Del mismo modo, el enlace se puede romper insertando un ordenador entre los dos dispositivos, formando un sistema *procesador de música*. En este caso la música no se representa como un patrón de variaciones de presión de aire (como el producido por un disco fonográfico o un disco compacto, en el que la música se codifica digitalmente), sino como una secuencia de *eventos* entrados en el teclado musical y transmitidos subsiguientemente al dispositivo de salida. La función de los circuitos para generación de sonido es la de convertir estos eventos en señales electrónicas que se reproducirán como sonido mediante un sistema de altavoz/amplificador. En consecuencia, la MIDI introduce por primera vez la seria perspectiva de controladores que no sean teclados, tales como instrumentos de cuerda o viento, siempre que sus respectivas salidas respondan a las especificaciones MIDI.

Hay un buen ejemplo de sistema para procesamiento de música que, aunque primario, es muy conocido, y se remonta a mucho antes de la edad del ordenador moderno. La pianola era un tipo especial de piano que podía tocar una música grabada en forma de patrón de agujeros en un rollo de papel. Mientras el papel rodaba a través del piano a una velocidad constante, un mecanismo detector captaba la secuencia y combinaciones de agujeros, cada uno de los cuales representaba una nota dada. Los martillos del piano golpeaban entonces las

Circuito del optoaislante

El segundo requisito del hardware MIDI especifica que la línea de entrada debe estar aislada del equipo recoptor para evitar bucles a tierra en el sistema. Éstos se deben evitar porque tienden a manifestarse en forma de un zumbido muy audible a la frecuencia de la red eléctrica (50 Hz). Este problema se puede superar mediante un dispositivo denominado *optoacoplador* u *optoaislante*. Se trata de un circuito integrado con una conexión óptica (y no eléctrica) entre su entrada y su salida (ver diagrama). Las dos patillas de entrada del chip optoaislante están conectadas a un LED interno, similar a las luces indicadoras rojas de la parte frontal de las unidades de disco. Un LED emite luz cuando se aplica un voltaje en la dirección positiva. En el optoacoplador el LED no está visible externamente, pero, en cambio, ilumina un fototransistor que permite que la corriente pase a través suyo cuando cae luz sobre él. La señal lógica (*low* o *high*) de la entrada se transfiere entonces a la salida sin ninguna conexión eléctrica. La MIDI especifica que el optoacoplador ha de necesitar una corriente de menos de 5 mA para encender el LED, y que los tiempos de ascenso y caída en la salida deben ser inferiores a dos microsegundos

Byte de datos en serie MIDI

Kevin Jones



cuerdas apropiadas. Así, la música se codificaba en el papel como una secuencia de eventos de pulsaciones de teclas.

La distancia de un agujero *a través* del papel representaba la altura de la nota, y la distancia *a lo largo* del papel representaba el momento en el cual había de tocarse la nota. ¡Era concebible, entonces, *editar* la música! Por ejemplo, si se perforaba una nota equivocada, se la podía localizar y tapar, mientras se perforaba en el rollo la nota correcta. La codificación de los datos de este modo tiene muchas analogías directas con los primeros sistemas de almacenamiento de datos en ordenador, que empleaban fichas perforadas.

Un procesador de música moderno se puede contemplar como una versión actualizada del sistema de pianola. El papel se reemplaza por la memoria del ordenador y el soporte de almacenamiento magnético, y el piano se reemplaza por un sintetizador con una interface de ordenador. Ahora la edición es simplemente cuestión de procesar los datos de la memoria del ordenador mediante un programa adecuado.

Pero antes de que podamos hacerlo (y, en realidad, incluso antes de que podamos introducir los datos en el ordenador) necesitamos saber dos cosas fundamentales acerca de la MIDI. La primera de éstas es el tipo de transmisión de datos utilizado y, la segunda, el formato de los datos. Estos dos factores corresponden a los componentes de hardware y software de las especificaciones de la interface; en este capítulo nos ocuparemos del hardware que se requiere para nuestro proyecto.

Transmisión de datos

Lo primero que necesitamos saber es cómo se transmite un byte de datos a través del enlace. La MIDI opera como una interface en serie asíncrona operando a 31.25 Kbaudios. Ello significa que sólo se puede transmitir un bit de datos a la vez, y que el receptor y el transmisor no están sincronizados entre sí mediante señales de reloj comunes. La desventaja de este método de transmisión de datos es su escasa velocidad. No obstante, puesto que el enlace se compone de apenas dos cables, exige cables y conectores muy sencillos y económicos.

La MIDI especifica que se han de utilizar los conectores estándares DIN de cinco patillas y 180° (con conectores profesionales de audio XLR de tres patillas como opción). Esto contribuye a mantener reducido el costo de la interface, favoreciendo su incorporación a instrumentos de precio económico.

La principal crítica de que es objeto la MIDI es, en realidad, su baja velocidad de transmisión en serie y, como veremos, las demoras de transmisión pueden volverse bastante notorias en un sistema grande. Sin embargo, en el lado positivo hay que destacar que la velocidad de transmisión es suficientemente lenta para la limitada velocidad de proceso que pueden afrontar los ordenadores personales.

Los datos MIDI se envían en grupos de 10 bits, cada uno de los cuales representa un byte de datos más un bit de comienzo y un bit de final. Cuando no se están enviando datos, la línea normalmente está establecida a un voltaje alto (+5 V para la MIDI). El inicio de una transmisión se señala mediante la puesta a *low* (0 V) de la línea durante un período de un bit (el bit de comienzo). Entonces se envían los ocho bits de datos (el MSB, bit más significativo, en primer lugar) seguidos por un período de un bit a un nivel más alto señalando el fin de los datos. Por consiguiente, la transmisión de cada byte lleva un total de 10 períodos de un bit, lo que equivale a 10/31.25 K=320 microsegundos. Como podremos comprobar más adelante en el proyecto, éste es un número muy importante que los aspirantes a programadores de MIDI deben tener presente.

Cuando la interface está inactiva, la línea se halla a un nivel de voltaje alto. El receptor inspecciona la línea continuamente atento a la primera aparición de un nivel bajo: el bit de comienzo de una transmisión. El receptor sabe entonces que se está enviando un byte de datos y puede establecer su contador, que cuenta períodos de un bit y medio desde la detección del bit de comienzo. Transcurrido este período, el receptor estará a medio camino del período de bit que contiene el MSB de los datos y, por tanto, puede detectar de forma fiable si el bit es un uno o un cero. Los bits restantes se reciben contando sucesivamente siete períodos de un bit y detectando cada vez el nivel.

Por último, se verifica el bit de detención en busca de un nivel alto, para comprobar que los datos estaban correctamente "delimitados" por los bits de comienzo y de final. Aplicando este método de comunicación, no es necesario que los relojes del receptor y el transmisor estén sincronizados con exactitud, simplificando aún más la interface. Sin embargo, la diferencia no debe ser tan acusada como para causar errores de temporización hacia el fin del byte y, por consiguiente, la MIDI especifica una tolerancia de un 1 % en la frecuencia estándar del reloj.

La interface es relativamente rápida, considerando su naturaleza en serie (la RS232 opera a una velocidad máxima de 19.2 Kbaudios) y, por tanto, los cables de conexión deben tener una longitud máxima de 15 m para evitar tiempos de ascenso y caída excesivamente lentos.

Demoras de transmisión

En este punto conviene analizar la limitación fundamental de la MIDI, a saber, las demoras de transmisión entre instrumentos. En general, un mensaje de nota encendida/apagada se compone de un total de tres bytes. El tiempo que consume enviar este mensaje es, en consecuencia, de 3×320 microsegundos=0.96 ms. Consideremos ahora la situación de un sistema de secuenciador de canales múltiples donde, pongamos por caso, puede ser necesario enviar 20 de estos mensajes a la vez. Esto significa que la última nota del grupo llega aproximadamente 20 ms más tarde de lo previsto.

Muchos secuenciadores que emplean grabación en tiempo real utilizan una resolución de temporización de tres o cuatro milisegundos para reproducir adecuadamente los matices de temporización de una actuación humana. Por tanto, una demora de 20 ms puede conducir a una temporización perceptiblemente deteriorada cuando se reproduce la grabación.

En el próximo capítulo analizaremos el software de las especificaciones MIDI.

El Nuevo Mundo (II)

He aquí la segunda parte del listado completo de nuestro juego de simulación mercantil

El juego está escrito en un BASIC mínimo y, por tanto, se podrá ejecutar en el BBC Micro con unas pocas alteraciones. Las principales modificaciones que es necesario introducir en el listado que ofrecemos atañen a la instrucción para limpiar la pantalla y al método de efectuar una pausa para la pulsación de una tecla. A lo largo del listado, allí donde aparezca la instrucción PRINT CHR$(147), los usuarios del BBC Micro han de reemplazarla por CLS.

El código utilizado para aguardar una pulsación de tecla en la versión de *El Nuevo Mundo* para el Commodore 64 (que es la versión que listamos aquí) implica establecer un bucle para tomar (GET) caracteres desde el teclado. El bucle termina cuando se recibe un carácter distinto a la serie nula. Este tipo de construcción aparece en muchos lugares del listado del programa, del siguiente modo:

```
<n.º línea>GET I$:IF I$="" THEN<n.º línea>
```

El BASIC BBC posee una instrucción especial con el mismo efecto que la anterior, pero más simple. Siempre que aparezca esta construcción, los usuarios del BBC deben reemplazarla por:

```
<n.º línea>I$=GET$
```

En el BBC Micro, las diferentes modalidades de visualización tienen diferentes requerimientos de memoria. Al objeto de dejar libre para el programa la máxima cantidad de memoria de BASIC, los usuarios del BBC Micro deben seleccionar la modalidad 7. Ésta es la modalidad que se selecciona automáticamente cuando se enciende el ordenador. Sin embargo, para asegurarse por completo, los usuarios de esta máquina quizá deseen añadir una línea extra al comienzo del programa:

```
5 MODE 7
```

```
5300 REM INFORME DE FINAL DE SEMANA
5305 PRINT CHR$(147)
5310 S$="                DIARIO DE NAVEGACION*":GOSUB 9100
5312 S$="                ------ -- ----------*      ":GOSUB 9100
5314 GOSUB 9200
5316 PRINT:PRINT"                FIN DE LA SEMANA";WK
5318 GOSUB 9200.
```

Este bucle comprueba si durante la semana ha fallecido algún tripulante, y reduce consiguientemente el número de tripulantes, CN.

```
5320 X=0
5325 FOR T=1 TO 16
5330 IF TS(T,2)<>-999 THEN 5350
5335 X=X+1
5340 TS(T,1)=0:TS(T,2)=0
5345 CN=CN-1
5350 NEXT
5355 IF X=0 THEN 5400
5358 PRINT:PRINT
5360 S$="DURANTE LA SEMANA PASADA*":GOSUB 9100
```

Si ha muerto toda la tripulación (lo que se indicaría mediante CN=0), el juego termina

```
5365 IF CN>0 THEN 5390
5367 S$="MURIO TODA LA TRIPULACION QUE QUEDABA*":GOSUB 9100
5369 GOSUB 9200:PRINT:GOSUB 9200
5373 S$="EL BARCO VA IRREMISIBLEMENTE A LA DERIVA*":GOSUB 9100
5374 GOSUB 9200
5376 S$="POR EL OCEANO DESIERTO......*":GOSUB 9100
5378 GOSUB 9200
5380 PRINT:PRINT
5382 S$="         EL JUEGO HA TERMINADO*":GOSUB 9100
5384 PRINT:PRINT
5386 GOSUB 9200:END
5388 GOTO 5386
5390 PRINT X;
5392 IF X=1 TEN S$= "MIEMBRO DE LA TRIPULACION HA FALLECIDO*"
5394 IF X<>1 THEN S$= "MIEMBROS DE LA TRIPULACION HAN FALLECIDO*"
5396 GOSUB 9100
5398 GOSUB 9200
5400 PRINT:PRINT
```

¿Se ha agotado alguna provisión? De ser así, se imprime un mensaje en tal sentido

```
5405 S$="AHORA TE HAS QUEDADO SIN "
5410 FOR T=1 TO 4
5415 IF PA(T)<>-999 THEN 5440
5420 PRINT S$;P$(T)
5425 PA(T)=0
5428 S$="                       Y"
5440 NEXT
5450 GOSUB 9200
5455 REM CALCULAR NUEVA JL
```

Se calcula el total de la fortaleza de la tripulación explorando TS(,) y, si resulta inferior a la mitad de la fortaleza total, se añade tiempo extra

```
5460 X=0
5465 FOR T=1 TO 16
5470 X=X+TS(T,2)
5475 NEXT
5480 IF X>799 THEN 5494
5481 PRINT:PRINT
5482 S$="LA TRIPULACION SE HALLA POR DEBAJO DE LA FORTALEZA
     TOTAL*":GOSUB 9100
5483 S$="DE MANERA QUE EL VIAJE PODRIA DURAR MAS*":GOSUB 9100
5489 EW=EW+((800-X)/800)
5490 JL=JL+INT(EW)
5492 PRINT
5494 S$=K$:GOSUB 9100
5496 GET I$:IF I$="" THEN 5496
5499 RETURN
```

Para generar eventos imprevisibles, se selecciona un número aleatorio y se utiliza para llamar una rutina mediante ON...GOTO

```
5500 REM GENERADOR DE EVENTOS AL AZAR
5502 IF RC=RM THEN RETURN
5503 REM SALIR SI TODOS LOS EVENTOS YA PRODUCIDOS
5504 PRINT CHR$(147)
5505 GOSUB 9200
5510 X=INT(RND(1)*RM)+1
5515 REM GENERAR NUM AL AZAR ENTRE 1 Y RM
5520 IF RR(X)=1 THEN RETURN
5522 REM RETURN SI ESTE EVENTO YA SE PRODUJO
5523 RR(X)=1:RC=RC+1
5524 REM ESTABLECER INDIC. PARA SEÑALAR QUE ESTE EVENTO YA SE
     PRODUJO E INCREMENTAR CONTADOR EVENTOS
5525 ON X GOTO 5540,5570,5570,5570,5570,5600,5700,5800,5850,5900,
     5950,6000,6050
5528 REM IR AL CODIGO APROPIADO PARA ESTE EVENTO
5530 PRINT:S$=K$:GOSUB 9100
5535 GET I$=IF I$="" THEN 5535
5539 RETURN:REM RETORNAR AL BUCLE PRINCIPAL VIAJE
```

Algunos eventos son nefastos...



```
5540 REM EVENTO 1-HOMBRE AL AGUA!
5542 PRINT
5543 S$=" DURANTE LA SEMANA*":GOSUB 9100
5545 PRINT:GOSUB 9200
5546 S$=" 1 PERSONA CAYO POR LA BORDA*":GOSUB 9100
5548 S$=" DURANTE UNA TORMENTA*":GOSUB 9100
5550 PRINT:GOSUB 9200
5552 S$="TU TRIPULACION SE HA REDUCIDO AHORA A*":GOSUB 9100
5554 PRINT CN-1;"MIEMBROS"
5558 FOR T=1 TO 16
5559 REM BUSCAR QUE TRIPULANTE SE PERDIO
5560 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 5566
5562 TS(T,2)=-999:REM MUERTO
5564 T=16
5566 NEXT
5568 GOTO 5530
5570 REM EVENTOS DEL 2 AL 5 - PERDIDA DE PROVISIONES
5572 X=X+1:REM AHORA X INDICA LA PROVISION(1-4)
5574 IF PA(X)=0 OR PA(X)=-999 THEN 5530
5576 REM NINGUNA ACCION SI YA SE HA AGOTADO ESTA PROVISION
5578 PRINT
5580 S$=" DURANTE LA SEMANA*":GOSUB 9100
5582 PRINT:GOSUB 9200
5584 PRINT" PARTE DE TU ";P$(X)
5586 S$=" FUE ARRASTRADA POR LA BORDA*":GOSUB 9100
5588 PRINT:GOSUB 9200
5590 S$="AHORA TIENES APROXIMADAMENTE*":GOSUB 9100
5592 PA(X)=PA(X)-INT(PA(X)/(INT(RND(1)*3)+2))
5593 REM REDUCIR CANTIDAD PROV EN 1/2 1/3 O 1/4
5594 PRINT INT(PA(X)/(CN*PN(X)));
5595 PRINT "Y TE QUEDA PARA ";P$(X);"SEMANAS APROXIMADAMENTE"
5599 GOTO 5530
```

...pero otros son beneficiosos. Por ejemplo, se puede sustituir con pescado la ración de carne

```
5600 REM EVENTO 6 - CAPTURAR PECES
5605 X=0
5610 FOR T=1 TO 16
5615 IF TS(T,2)=-999 THEN X=X+1
5620 REM CONTAR FALLECIDOS ESTA SEMANA
5625 NEXT
5630 IF CN-X<1 THEN RETURN
5635 REM NINGUNA ACCION SI TODA TRIPULACION MUERTA
5640 PRINT
5645 S$=" DURANTE LA SEMANA*":GOSUB 9100
5646 PRINT:GOSUB 9200
5650 X$="UNO DE LOS TRIPULANTES CAPTURO*":GOSUB 9100
5655 X=INT(RND(1)*10)+11
5660 REM ENTRE 10 y 20 KILOS
5662 PRINT X;"KILOS DE PESCADO"
5665 PRINT:GOSUB 9200
5670 S$="AHORA TU PROVISION DE CARNE ES*":GOSUB 9100
5675 S$="SUFICIENTE PARA APROXIMADAMENTE*":GOSUB 9100
5678 IF PA(3)=-999 THEN PA(3)=0
5680 PA(3)=PA(3)+X
5685 PRINT INT(PA(3)/(CN*PN(3)));"SEMANAS"
5690 GOTO 5530
```

Si se estuviera acabando el agua, esta rutina permitirá el reabastecimiento. En la línea 5735 se genera al azar un número entre 10 y 20, que representa los barriles de agua que se añadirán al stock retenido en PA(4)

```
5700 REM EVENTO 7 - RECOGER AGUA
5705 PRINT
5710 S$=" DURANTE LA SEMANA*":GOSUB 9100
5715 PRINT:GOSUB 9200
5720 S$="UN TEMPORAL DE LLUVIA LLENO TUS*":GOSUB 9100
5725 S$="BARRILES DE AGUA*":GOSUB 9100
5730 PRINT:GOSUB 9200
5735 X=INT(RND(1)*10)+11
5736 REM ENTRE 10 y 20 BARRILES
5740 S$="AHORA TU PROVISION DE AGUA ES*":GOSUB 9100
5745 S$="SUFICIENTE PARA APROXIMADAMENTE*":GOSUB 9100
5748 IF PA(4)=-999 THEN PA(4)=0
5750 PA(4)=PA(4)+X
5755 PRINT INT(PA(4)/(CN*PN(4)));"SEMANAS"
5760 GOTO 5530
5800 REM EVENTO 8 - VIENTOS FAVORABLES
5805 PRINT
5810 S$="FUERTES VIENTOS CONTINUOS DURANTE TODA LA
     SEMANA*":GOSUB 9100
5815 PRINT:GOSUB 9200
5820 S$="HAS NAVEGADO A BUENA VELOCIDAD*":GOSUB 9100
5825 S$="Y LA DURACION DEL VIAJE SE *":GOSUB 9100
5830 S$="REDUCE EN MEDIA SEMANA*":GOSUB 9100
5835 EW=EW-.5
5839 GOTO 5530
```

El buen tiempo puede mejorar la fortaleza de la tripulación. Esto se consigue explorando la matriz (TS(,)) e incrementando los coeficientes individuales en función de un número entre 5 y 15

```
5850 REM EVENTO 9 - BUEN TIEMPO
5855 PRINT
5860 S$="BUEN TIEMPO DURANTE TODA LA SEMANA*":GOSUB 9100
5865 PRINT:GOSUB 9200
5870 S$="LA TRIPULACION SE SIENTE MAS FELIZ*":GOSUB 9100
5875 GOSUB 9200
5880 S$="          Y ESTA MAS SALUDABLE!*":GOSUB 9100
5882 FOR T=1 TO 16
5884 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 5888
5886 TS(T,2)=TS(T,2)+INT(RND(1)*11)+5
5888 NEXT
5889 GOTO 5530
```

Las tres rutinas siguientes pueden, al ser llamadas, disminuir el cargamento al estropearse las armas, las balas de tela y los frascos

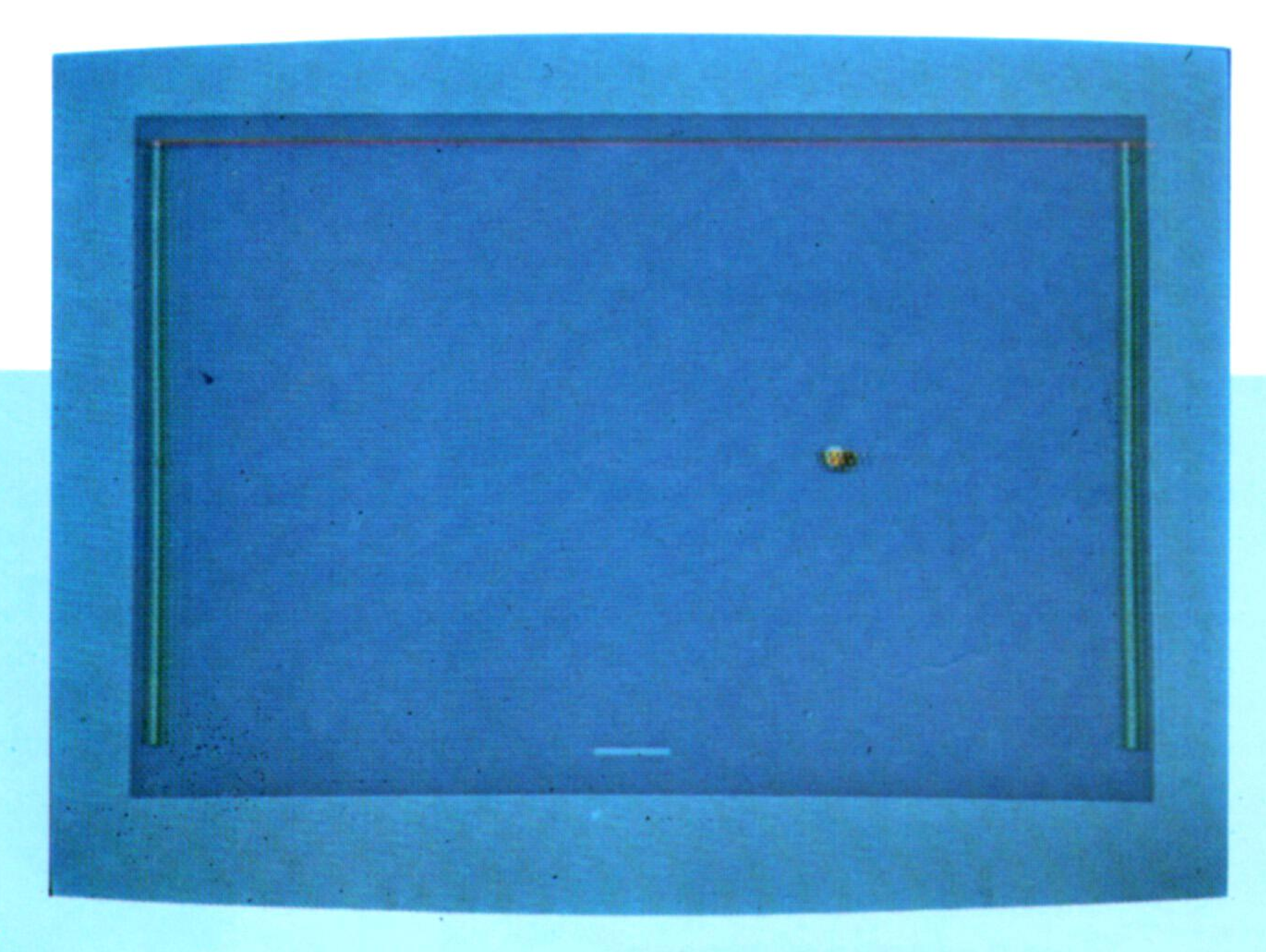
```
5900 REM EVENTO 10 - PERDIDA DE MEDICINAS
5905 IF OA(1)=0 OR OA(1)=-999 THEN RETURN
5910 PRINT
5915 S$="DESCUBRES QUE LA MITAD DE TUS*":GOSUB 9100
5920 S$="FRASCOS DE MEDICINA SE HAN ROTO*":GOSUB 9100
5925 OA(1)=INT(OA(1)/2)
5930 PRINT:GOSUB 9200
5935 S$="AHORA TE QUEDAN*":GOSUB 9100
5940 PRINT OA(1);"FRASCOS SOLAMENTE"
5945 GOTO 5530
5950 REM EVENTO 11 - ARMAS OXIDADAS
5955 IF OA(2)=0 OR OA(2)=-999 THEN RETURN
5960 PRINT
5965 S$="DESCUBRES QUE LA MITAD DE TUS*":GOSUB 9100
5970 S$="ARMAS SE HAN OXIDADO*":GOSUB 9100
5972 S$="Y YA NO SIRVEN*":GOSUB 9100
5975 OA(2)=INT(OA(2)/2)
5980 PRINT:GOSUB 9200
5985 S$="AHORA TE QUEDAN*":GOSUB 9100
5990 PRINT OA(2);"ARMAS SOLAMENTE"
5995 GOTO 5530
6000 REM EVENTO 12 - PERDIDA DE TELA
6005 IF OA(4)=0 OR OA(4)=-999 THEN RETURN
6010 PRINT
6015 S$="DESCUBRES QUE LOS RATONES*":GOSUB 9100
6020 S$="SE HAN COMIDO LA MITAD DE TUS*":GOSUB 9100
6022 S$="BALAS DE TELA Y AHORA*":GOSUB 9100
6024 S$="ESTAS YA NO SIRVEN*":GOSUB 9100
6025 OA(4)=INT(OA(4)/2)
6030 PRINT:GOSUB 9200
6035 S$="AHORA TE QUEDAN*":GOSUB 9100
6040 PRINT OA(4);"BALAS SOLAMENTE"
6045 GOTO 5530
6050 REM EVENTO 13 - ALBATROS
6055 PRINT:A$="S"
6060 S$="UN ALBATROS SOBREVUELA EL BARCO*":GOSUB 9100
6062 GOSUB 9200
6065 S$="ESTE ES UN BUEN PRESAGIO*":GOSUB 9100
6068 S$="Y LA TRIPULACION ESTA CONTENTA*":GOSUB 9100
6070 PRINT:GOSUB 9200
6075 IF PA(3)<(CN*PN(3)*(JL-WK+1)) THEN 6090
6080 REM NO HAY ESCASEZ DE CARNE
6085 GOTO 6122
6090 S$="SE TE ESTA TERMINANDO LA CARNE*":GOSUB 9100
6095 S$="Y EL AVE PESA 10 KILOS!*":GOSUB 9100
6100 PRINT:GOSUB 9200
6105 S$="TE GUSTARIA COGERLA?*":GOSUB 9100
6110 INPUT I$
6112 PRINT:GOSUB 9200
6115 IF LEFT$(I$,1)="S" THEN 6133
6120 S$="PROBABLEMENTE DA LO MISMO!*":GOSUB 9100
6122 PRINT:GOSUB 9200
6125 S$="EL ALBATROS SE ALEJA VOLANDO......*":GOSUB 9100
6130 GOTO 5530
6133 IF OA(2)=0 OR OA(2)=-999 THEN 6180
6135 S$="HACES UN DISPARO.........*":GOSUB 9100
6138 GOSUB 9200:GOSUB 9200
6140 IF RND(1)<.5 THEN 6150
6145 S$=".........PERO NO ACIERTAS!*":GOSUB 9100
6148 GOTO 6122
6150 S$="Y EL AVE CAE SOBRE LA CUBIERTA!*":GOSUB 9100
6155 PRINT:GOSUB 9200
6160 IF PA(3)=-999 THEN PA(3)=0
6162 PA(3)=PA(3)+10:B$="S"
6165 S$="AHORA TIENES 10 KILOS MAS*":GOSUB 9100
6167 S$="DE CARNE .........*":GOSUB 9100
6170 S$="PERO PUEDE SER QUE LA BUENA FORTUNA*":GOSUB 9100
6172 S$="NO TE ACOMPAÑE A PARTIR DE AHORA!!*":GOSUB 9100
6174 GOTO 5530
6180 S$="NO PUEDES - NO TIENES ARMAS*":GOSUB 9100
6190 GOTO 6122
```

Los eventos ya reseñados poseen una mínima influencia sobre el viaje, pero también se pueden producir otros más determinantes, seleccionados al azar mediante esta rutina

```
6500 REM EVENTOS MAYORES
6508 X=INT(RND(1)*10)+1
6510 ON X GOSUB 6530,6700,6800,6900,7000,7050
6520 RETURN
```

Squash en el C64

Un juego deportivo no hace mal a nadie. Éste le ofrece la posibilidad de practicarlo si es usuario de un Commodore 64



La raqueta se desplaza con ayuda de las teclas de control del cursor (a la derecha de la barra espaciadora). Dispone de diez pelotas que ha de mantener en juegó el mayor tiempo posible. Cada pelota que se devuelve proporciona un punto.

```
5 REM ************
10 REM *   SQUASH   *
15 REM ************
20 GOTO 1000
50 V=V+DV
60 H=H+DH
70 POKE M+B,BN
80 B=V*X+H
90 POKE M+B,BO
100 POKE N+B,C
110 IF V=V1 AND ABS(H-(POS(X)+3))>1 THEN 2000
120 IF V=V1 OR V=V0 THEN DV=-DV
130 IF H>=H1 OR H<=H0 THEN DH=-DH
140 IF V=V1 THEN S=S+1
150 GET X$
160 IF X$=C$ AND POS(X)<>0 THEN PRINT GG$;:IF V=V1
    THEN H=H-1
170 IF X$=D$ AND POS(X)<>34 THEN PRINT DD$;:IF V=V1
    THEN H=H+1
180 PRINT R$;
190 PRINT T$;
200 GOTO 50
1000 X=40
1010 M=1024
1020 N=55296
1030 D$=CHR$(29)
1040 G$=CHR$(17)
1050 R0=0
1060 R1=33
1070 BN=32
1080 B=0
1090 C=8
1100 B0=81
1110 H=INT(RND(TI)*(X-4)+2)
1120 V=22
1130 H0=2
1140 H1=X-2
1150 V0=1
1160 V1=22
1170 DV=-1
1180 DH=(RND(TI)<0.5)*2+1
1190 R$=CHR$(32)+CHR$(32)+CHR$(196)
    +CHR$(196)+CHR$(196)+CHR$(32)+CHR$(32)
1200 R=17
1220 B=V*X+H
1230 IF NB=0 THEN GOSUB 1500
1240 GOTO 50
1500 PRINT CHR$(147);
1510 FOR I=0 TO 40
1520 POKE M+I,100
1530 POKE N+I,2
1540 NEXT I
1550 FOR I=1 TO 22
1560 POKE M+I*40,103
1570 POKE N+I*40,5
1580 POKE M+I*40+39,101
1590 POKE N+I*40+39,5
1600 NEXT I
1610 DD$=CHR$(29)+CHR$(29)
1620 GG$=CHR$(157)+CHR$(157)
1630 T$=""
1640 FOR I=1 TO 7
1650 T$=T$+CHR$(157)
1660 NEXT I
1670 PRINT CHR$(19)
1680 FOR I=1 TO 22
1690 PRINT G$;
1700 NEXT I
1710 FOR I=1 TO 16
1720 PRINT CHR$(32);
1730 NEXT I
1740 S=0
2000 NB=NB+1
2010 POKE M+B,BN
2020 POKE 54296,15
2030 POKE 54277,190
2040 POKE 57278,136
2050 POKE 54273,17
2060 POKE 54272,37
2070 POKE 54276,65
2080 FOR I=1 TO 50
2090 NEXT I
2100 POKE 54272,0
2110 POKE 54273,0
2120 POKE 54276,0
2130 FOR I=1 TO 500
2140 NEXT I
2150 IF NB=11 THEN 3000
2160 GOTO 1000
3000 IF S>RR THEN RR=S
3010 NB=0
3020 PRINT CHR$(147);
3030 FOR I=1 TO 7
3040 PRINT
3050 NEXT I
3060 PRINT TAB(16)"PUNTOS[1SPC]:";
3070 PRINT S
3080 FOR I=1 TO 4
3090 PRINT
3100 NEXT I
3110 PRINT TAB(12)"PUNTUACION[1SPC]MAXI
    MA[1SPC]:";
3120 PRINT RR
3130 FOR I=1 TO 4
3140 PRINT
3150 GET X$
3155 GET X$
3160 NEXT I
3170 PRINT TAB(16)"OTRA[1SPC]?"
3180 GET X$
3190 IF X$="" THEN 3180
3200 IF X$<>"N" THEN 20
3210 END
```


E R T Y U I O
D F G H J K L
X C V B N M

10091
9 788485 822836